НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.9

«До захисту допущено»

Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)

"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Спеціалізовані комп'ютерні системи

на тему: Способи передачі інформації по лініії живлення

Виконала: студентка II курсу, групи _КВ-73мп_____
(шифр групи)

Максим Катерина Євгенівна_____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник_к.т.н._доцент каф. СПСКС_Боярінова Ю. Є._ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Спеціалізовані комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ _В.П.Тарасенко__
(підпис) (ініціали, прізвище)
«___»_____2018р.

ЗАВДАННЯ
на магістерську дисертацію студентки
Максим Катерини Євгенівни

1. Тема дисертації: СПОСОБИ ПЕРЕДАЧІ ІНФОРМАЦІЇ ПО ЛІНІЇ ЖИВЛЕННЯ,
науковий керівник дисертації: к.т.н., доцент каф.СПСКС Боярінова Ю.Є,
затверджені наказом по університету від «30» жовтня 2018 р. № 4030-с.

2. Термін подання студентом дисертації: 10 грудня 2018 р.

3. Об'єкт дослідження: методи та алгоритми передачі даних.

4. Предмет дослідження: є методи оптимізації обміну даними по лінії живлення в багаторівневих системах котролю.
5. Перелік завдань, які потрібно розробити:
- розглянути існуючі методи організації топології мереж для передачі даних в багаторівневих системах контролю;
- провести порівняльний аналіз основних методів передачі даних;
- дослідити методи передачі даних по лінії живлення;
- розробити та описати етапи обраного алгоритму на основі реалізації програмного UARTпрограмно реалізувати розроблений алгоритм передачі даних;
- провести тести і оцінити роботу алгоритму.

6. Перелік ілюстративного матеріалу:
   - Презентація.

7. Перелік публікацій: система передачі даних в багаторівневих системах контролю була представлена та обговорювалась на науковій конференції магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018 (Київ, 14-16 листопада 2018 р.) ; міжнародна наукова конференція «Системний аналіз та інформаційні технології» SAIT – 2018 (Київ, 21-23 травня 2018).

8. Дата видачі завдання 10 вересня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Ознайомлення з предметною галуззю	18.10.2017	
2	Вивчення літератури за тематикою магістерської дисертації, пошук додаткової літератури	07.02.2018	
3	Робота над першим розділом магістерської дисертації; огляд існуючих рішень; робота над публікацією для міжнародної конференції «SAIT-2018»	21.03.2018	
4	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; пошук технологій для реалізації програмного забезпечення	27.06.2018	
5	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	12.09.2018	
6	Проведення наукового дослідження; розробка програмного забезпечення; підготовка матеріалів тез доповіді	05.10.2018	
7	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	16.10.2018	
8	Оформлення текстової і графічної частини магістерської дисертації	10.11.2018	
9	Попередній розгляд магістерської дисертації на кафедрі	26.11.2018	

Студент _____ Максим К. Є.

Науковий керівник дисертації _____ Боярінова Ю.Є.

РЕФЕРАТ

Актуальність теми.

З розвитком таких технологій, як розумний будинок виникає необхідність в надійних технологіях передачі даних. Подібні системи складаються з багатьох пристроїв контролю та обробки інформації. Забезпечення надійності приладів і цілісності ліній зв'язку є основною умовою безвідмовності роботи системи. Адже пожежі можуть траплятись і у оснащених пожежною сигналізацією об'єктах. Такі випадки трапляються доволі рідко, але мають місце серед статистики пожеж. Факторами неспрацювання пожежної сигналізації можуть бути як і людський, так і технічний фактори. До людських факторів може привести ненадійне встановлення дротових зв'язків між пристроями пожежної сигналізації що призвело до погіршення якості з'єднання, або до обриву чи замикання лінії. Технічними факторами можуть бути вихід з ладу будь-якого компонента, електромагнітні завади, тощо. Навіть застосування сучасних технологій не гарантує повну безпеку від виникнення пожежі, але для мінімізації її наслідків дуже важливим є своєчасне виявлення пожежі та оповіщення про неї. Тому дослідження методів підвищення надійності системи пожежної безпеки, автоматизації контролю, інтелектуальної системи прийняття рішень та методів обміну інформацією по лінії живлення є актуальним.

Об'єктом дослідження є методи сполучення пристроїв в одній системі для передачі інформації в багаторівневих системах контролю.

Предметом дослідження є методи оптимізації обміну даними по лінії живлення.

Мета роботи: пошук оптимальних методів підвищення стійкості ліній зв'язку багаторівневих систем та методів оптимізації зв'язку між пристроями.

Наукова новизна:

1. В даній роботі розглянуті методи підвищення стійкості ліній зв'язку багаторівневих систем та запропоновано інноваційний підхід до оптимізації

зв’язку між пристроями що спрощує налаштування та зменшує затрати на кабельно-провідникові матеріали.

Практична цінність отриманих в роботі результатів полягає в тому, що розроблений модифікований спосіб дозволяє спростити організацію мережі і підключення систем.

Інноваційний метод сполучення пристроїв дозволяє забезпечити надійний зв'язок приладів в одній системі.

Апробація роботи. Система передачі даних в багаторівневих системах контролю була представлена та обговорювалась на науковій конференції магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2018 (Київ, 14-16 листопада 2018 р.) ; міжнародна наукова конференція «Системний аналіз та інформаційні технології» SAIT – 2018 (Київ, 21-23 травня 2018).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано узагальнену оцінку сучасного стану проблеми, обґрунтовано актуальність виконаного дослідження, дано загальну характеристику роботи, поставлено мету та задачу дослідження, і наведено практичну цінність роботи.

У першому розділі розглянуто існуючі методи передачі даних по лініям постійної та змінної напруги та через радіо канали, їх принципи роботи та особливості, недоліки та переваги. Розглянуто різні існуючі методи , що вирішують дану проблему.

У другому розділі розглянуто технології, що застосовані для реалізації розробки.

У третьому розділі описано деталі реалізації розробленої системи.

У четвертому розділі описано тестування розробленої системи.

У висновках представлені результати проведеної роботи.

Робота представлена на 82 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: багаторівневі системи контролю, лінія живлення.

ABSTRACT

Actuality of theme.

With the development of technologies such as smart home, there is a need for reliable data transfer technologies. Such systems consist of many control and information processing devices. Ensuring the reliability of devices and the integrity of the communication lines is a basic condition for the system's failure-free operation. After all, fires can also occur in objects equipped with fire alarm. Such cases occur quite rarely, but occur among the statistics of fires. Factors that do not trigger a fire alarm can be both human and technical. Human factors may be caused by the unreliable installation of wired communications between fire alarm devices, which led to poor connection quality, or to the breakdown or closure of the line. Technical factors may be the failure of any component, electromagnetic interference, and the like. Even the use of modern technology does not guarantee full safety from the fire, but to minimize its consequences, it is very important to timely detect the fire and notify it. Therefore, the study of methods for increasing the reliability of the fire safety system, automation of control, intellectual decision-making system and methods of information exchange on the power supply is relevant.

The object of the study is the methods of combining devices in one system for the transmission of information in multi-level control systems.

The subject of the study is the methods of optimizing the data exchange by power supply.

The purpose of the work: the search for optimal methods for increasing the stability of communication lines of multi-level systems and methods for optimizing communications between devices.

Scientific novelty:

1. In this paper, methods of increasing the stability of the communication lines of multilevel systems are considered, and an innovative approach to optimizing communication between devices that simplifies the configuration and reduces the cost of cable-conductor materials is proposed.

The practical value of the results obtained in the work is that the developed modified method allows simplifying the organization of the network and connection of systems. Innovative method of connecting devices allows to ensure reliable communication of devices in one system.

Test work. The data transmission system in the multilevel control systems was presented and discussed at the scientific conference of masters and postgraduates "Applied Mathematics and Computer", PMK-2018 (Kyiv, November 14-16, 2018); International Scientific Conference "System Analysis and Information Technologies" SAIT-2018 (Kyiv, May 21-23, 2018).

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The *introduction* provides a generalized assessment of the current state of the problem, substantiates the relevance of the research, gives a general description of the work, sets the goal and purpose of the research, and gives practical value to the work.

The first chapter examines the existing methods of data transmission over the lines of constant and alternating voltage and through radio channels, their principles of operation and features, disadvantages and advantages. Different existing methods that solve this problem are considered.

The second section discusses the technologies used to implement the development.

The third section describes the details of the implementation of the developed system.

In the fourth section, we present the methods of testing the system and compare it with the analogues.

In the conclusions are the results of the work.

The work is presented on 82 sheets, contains a link to the list of used literary sources.

Key words: multilevel control systems, power line.

ЗМІСТ

ВСТУП

З розвитком сучасних технологій та багаторівневих систем зв’язку, є необхідність в надійних методах передачі даних. Однією з таких є обмін даними по лінії живлення.

Найочевидніша перевага використання високовольтних комунікацій для передачі даних - відсутність необхідності прокладати кабелів і здійснення монтажних робіт. Стандартизація широкосмугової передачі по високовольтних електромережах надає більш гнучку, надійну і вигідну альтернативу традиційним технологіям для побудови високошвидкісних мереж передачі даних в будівлях.

Для вітчизняних розробників найбільш перспективними галузями застосування даного виду зв'язку є системи віддаленого збору даних з лічильників, охоронні системи, системи типу «розумний дім».

Основну увагу зосереджено на забезпеченні надійності зв’язку та огляді мереж в системах безпеки та сигналізації. Часто в системах безпеки використовують найбільш прості і нестійкі до обривів або коротких замикань топології мереж. Це виправдано підходом замовників, коли ціна вирішує все. Але в системах безпеки це недопустимо і головним завданням є не просто запустити в роботу систему, а й мінімізувати можливі втрати в її роботі від зовнішніх впливів на обладнання та мережі в процесі експлуатації. Знання особливостей систем допоможе зробити системи інструментом забезпечення безпеки, а не набором устаткування, яке при першій же дії (через недбалість або зловмисну дію) перетворитися в купу металобрухту.

Для створення надійної системи бажано, а при наявності відповідних нормативних документів необхідно, використовувати топологію мережі, що забезпечує безперервну роботу в разі зловмисного або випадкового пошкодження зв'язків з компонентами системи.

Система, що відповідає за безпеку має забезпечувати безвідмовну роботу навіть у критичних випадках. Будь то вибух чи вогонь що можуть вплинути на будь-яку частину системи і вивести її з ладу чи некоректні умови

експлуатації пристроїв інша частина системи не повинна перейти у несправний стан. Резервація компонентів повинна забезпечити безвідмовну роботу системи при надзвичайних випадках. Критичними до резервації в першу чергу є лінії підключення.

В даній роботі розглянуті методи підвищення стійкості ліній зв’язку багаторівневих систем та запропоновано інноваційний підхід до оптимізації зв’язку між пристроями що спрощує налаштування та зменшує затрати на кабельно-провідникові матеріали.

1.ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОРГАНІЗАЦІЇ ПЕРЕДАЧІ ДАНИХ В БАГАТОРІВНЕВИХ СИСТЕМАХ КОТРОЛЮ

1.1 Огляд топологій мережі для системи безпеки.

Часто в системах безпеки використовують найбільш прості і нестійкі до обривів або коротких замикань топології мереж. Це виправдано підходом замовників, коли ціна вирішує все. Але в системах безпеки це недопустимо і головним завданням є не просто запустити в роботу систему, а й мінімізувати можливі втрати в її роботі від зовнішніх впливів на обладнання та мережі в процесі експлуатації. Знання особливостей систем допоможе зробити системи інструментом забезпечення безпеки, а не набором устаткування, яке при першій же дії (через недбалість або зловмисну дію) перетворитися в купу металобрухту.

Для створення надійної системи безпеки бажано, а при наявності відповідних нормативних документів необхідно, використовувати топологію мережі, що забезпечує безперервну роботу в разі зловмисного або випадкового пошкодження зв'язків з компонентами системи. Розглянемо топології і особливості мереж для безпеки. На рис. 1.1 показані типи топологій для централізованих систем.

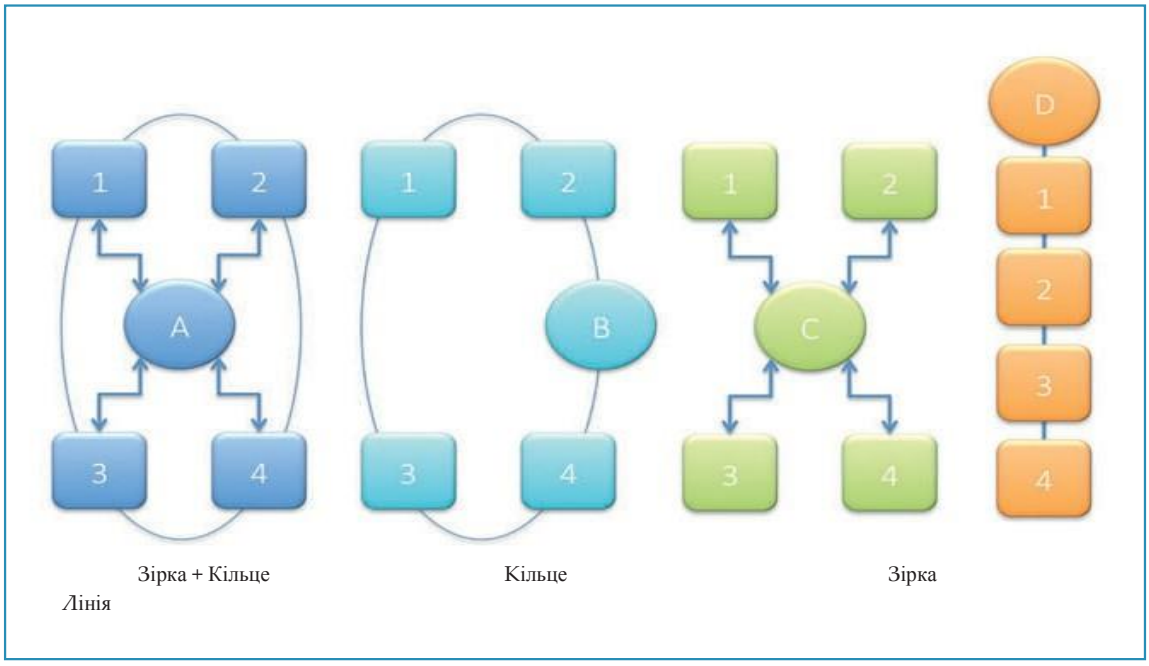


Рисунок 1.1 - Варіанти топології мережі централізованої системи

Варіанти розташовані зліва направо - A, B, C, D в порядку убування «живучості» мережі. Підключення приладів по топології A («зірка плюс кільце») зберігає зв'язок з компонентом при двох коротких замиканнях або обривах зв'язку ліній, підключених до цього компоненту. Важливо, що всі використовувані для створення такої топології мережі чотири входи приладу повинні бути незалежними і мати захист від КЗ. Підключення приладів по топології B («кільце») дозволяє зберегти працездатність мережі при одному короткому замиканні або обриві в лінії кільця. Обидва входи (або вхід / вихід) центрального приладу повинні бути незалежними і мати захист від КЗ. При використанні топології C («зірка») зв'язок буде втрачено лише в пошкодженій лінії, якщо всі входи центрального приладу незалежні і мають захист від КЗ. Якщо прилади в системі підключені по топології D («лінія»), то коротке замикання в будь-якому місці лінії призведе до припинення зв'язку з усіма компонентами ланцюга. Незважаючи на настільки очевидну недосконаліть топології D на практиці вона використовується найбільш часто. Для децентралізованих систем мережі можуть мати топологію, показану на рис. 1.2.

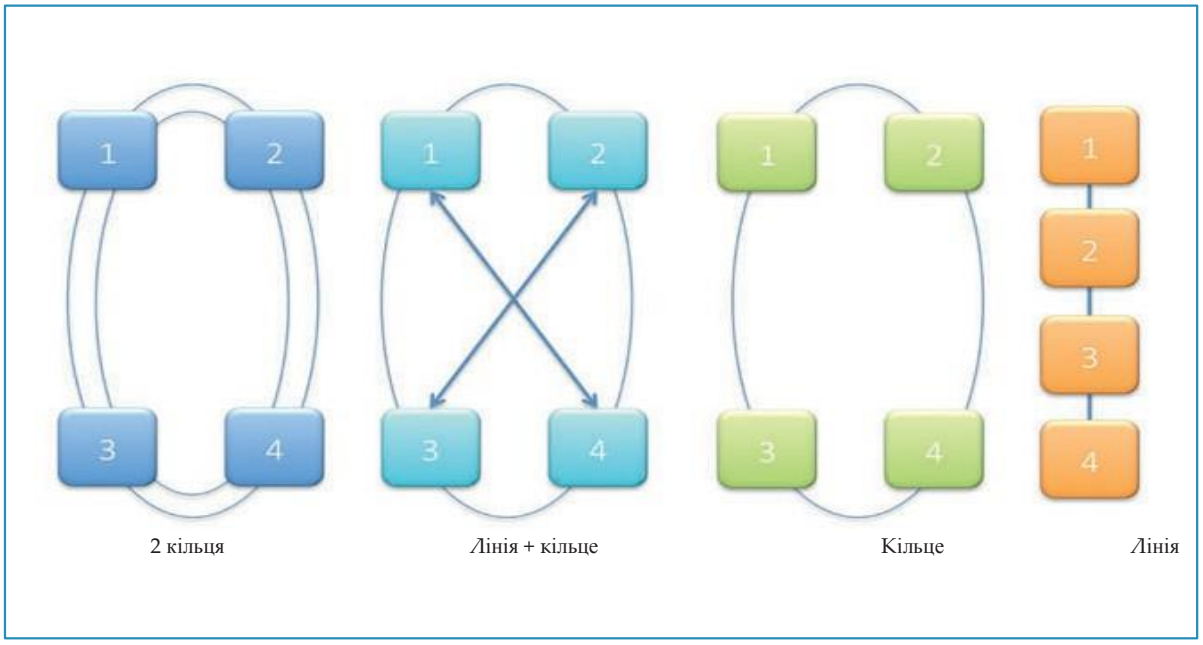


Рисунок 1.2. - Варіанти топології мережі децентралізованої системи

Знову ж, топологія «лінія» використовується на практиці найбільш часто, незважаючи на те, що коротке замикання в лінії може привести до припинення

зв'язку з усіма компонентами мережі. Інші варіанти більш стійкі до короткого замикання або обриву лінії. Підключення приладів по топології «кільце» дозволяє зберегти працездатність мережі при одному короткому замиканні. Якщо прилади в системі підключені по топології «лінії плюс кільце», то зв'язок між компонентами зберігається при двох коротких замиканнях. При використанні топології «2 кільця» працездатність мережі зберігається при трьох коротких замиканнях. Для кільцевої топології самовідновлення є звичною властивістю. У більшості випадків лінійна частина кільцевої структури будується на основі пари дротів (так зване здвоєне кільце). В результаті у передавального вузла є два способи отримати доступ до приймального: за годинниковою стрілкою і в зворотному напрямку. Один з маршрутів виконує функції основного і використовується для передачі даних, інший розглядається як резервний.

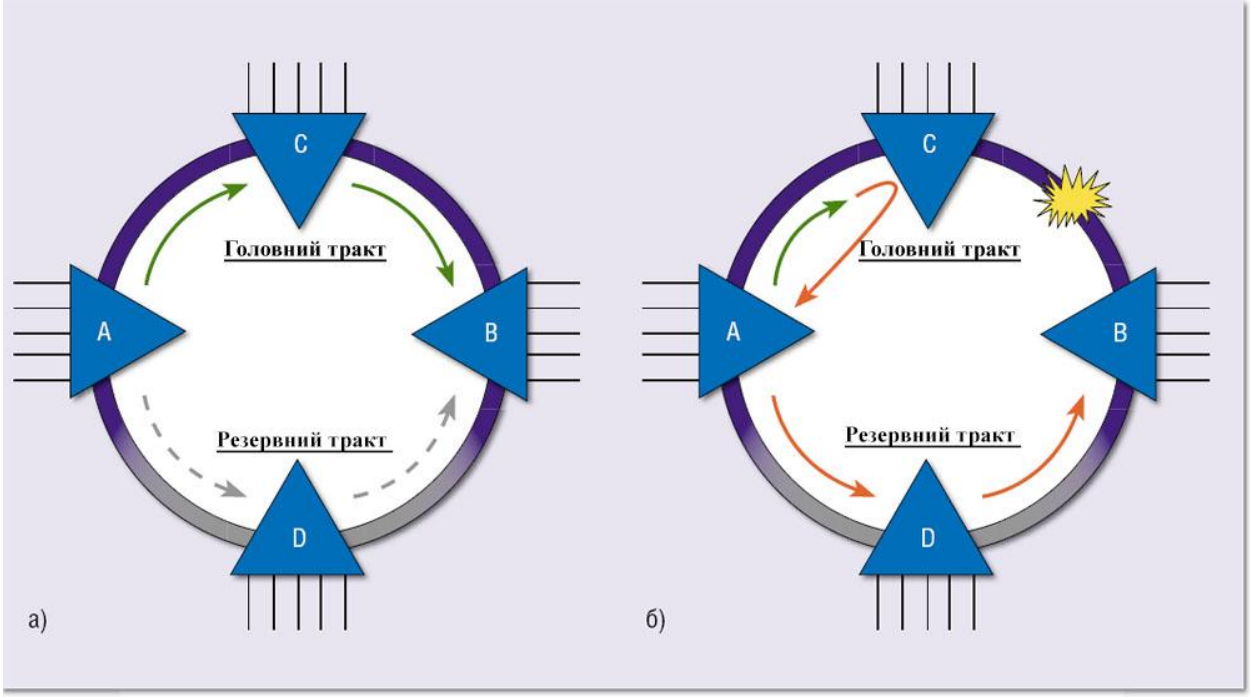


Рисунок 1.3 - Мережа з кільцевою топологією

Мережа з кільцевою топологією відкриває можливість для використання різних схем резервування, що значно підвищує надійність функціонування систем зв’язку.

Організація резервування в кільцевій топології не вимагає значних витрат на збільшення кількості дротів або прокладку додаткових кабелів. Це

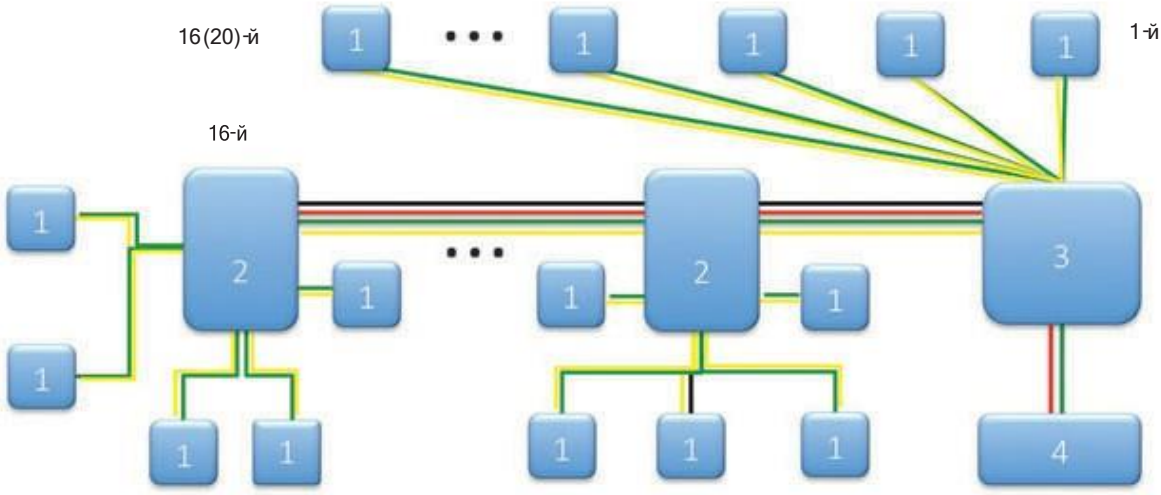
дає значну перевагу виходячи з матеріальної точки зору, адже в системах пожежної безпеки для підключення пристроїв використовують спеціальні жаростійкі дроти, що мають набагато більшу вартість а ніж звичайні UTP чи FTP. Це дає можливість значно економити матеріал використовуючи саме кільцевий інтерфейс.

Історично першими типами підключення периферійного обладнання до керуючих приладів / контролерам були прямі підключення. Один сповіщувач або виконавець - одна лінія зв'язку. У такій схемі багато проводів, але вихід з ладу однієї лінії ніяк не впливає на роботу інших. Керуючі прилади / контролери з'єднувалися з центром моніторингу по телефонних лініях. То була епоха одиничних локальних мереж, коли Інтернету ще не існувало. З поліпшенням технологій зв'язку і зростанням кількості компонентів змінювалися схеми підключення. Розвиток відбувався в двох напрямках:

1. Підключення до контролерів периферійних пристроїв.
2. З'єднання між собою контролерів і підключення до серверів.

Розглянемо ці два напрямки і спочатку зупинимося на підключенні контролерів до периферійних пристроїв. Наступним кроком після прямого підключення стало приєднання до контролерів периферійних пристроїв за допомогою розширювальних модулів / блоків, які з'єднувалися з виділеними групами клем контролерів. Периферійне устаткування як і раніше підключалося до пари клем такого модуля лінією з двох проводів. Живлення периферійних пристроїв надходило або від розширювального модуля або від контролера. Потім настала черга використання розширювальних модулів, з'єднаних з контролером по адресній шині (рис. 1.4). Логічним продовженням такого шляху було збільшення кількості адрес і використання кожного адресного модуля для підключення одного периферійного пристрою. Так з'явилися адресні шини та адресні прилади. Так з'явилися системи, де кожний периферійний пристрій має свою адресу і підключається до керуючого приладу по лінії зв'язку, що подальшому буде мати назву «шина». Тут слід зупинитися на кількості адрес в одній шині. В силу особливостей протоколу і

підходів до співвідношення надійність / ціна виробники почали створювати прилади з кількістю адрес в шині: 32 (50), 127, 254, 512. Цей ряд можна продовжити, але очевидно, що чим більше адрес в одній шині тим менше надійність такого з'єднання.



1 – неадресний пристрій
2 – адресний модуль розширення
3 – контролер
4 – джерело живлення

Рисунок 1.4 - Підключення розширювальних модулів до контролера по адресній шині

1.2 Огляд рішень для передачі даних методами бездротового зв’язку.

Розглядаючи рішення для передачі даних побудовані на бездротовій мережі зв’язку слід виділити основні вимоги до таких мереж. Бездротова мережа зв’язку повинна бути надійною, здатною до самовідновлення, простою в розгортанні та експлуатації. Важливо також, щоб обладнання таких мереж допускало тривалу роботу від автономних джерел живлення, мало низьку вартість, і було компактним. Такі вимоги є основними для функціонування таких систем як «розумний дім», систем пожежної та охоронної сигналізації.

Зазвичай бездротові мережі ділять на категорії:

- частково бездротова - закладається на етапі ремонту. Блоки живляться від силової проводки, а взаємодія компонентів відбувається через Wi-Fi.
- повністю бездротова - її можна проводити вже після ремонту. Ця система являє собою набір пристроїв, які можна самостійно синхронізувати між собою за допомогою інтернету.

Бездротові системи спрощують встановлення та організацію приладів. Завдяки можливості ретранслювання відстані можуть бути розширені в десятки разів, у порівнянні зі звичайним Wi-Fi [1].

На рис.1.5 показана мережа типу "зірка" в порівнянні з сітчастою мережею. В мережі "зірка" (Wi-Fi) весь трафік прямує через центральну точку, а в сітчастій мережі (ZigBee, Thread, Z-wave) зв'язок забезпечується через сусідні точки. Кожен з цих методів має свої особливості, і вибір правильного поєднання - це важлива частина процесу розробки.

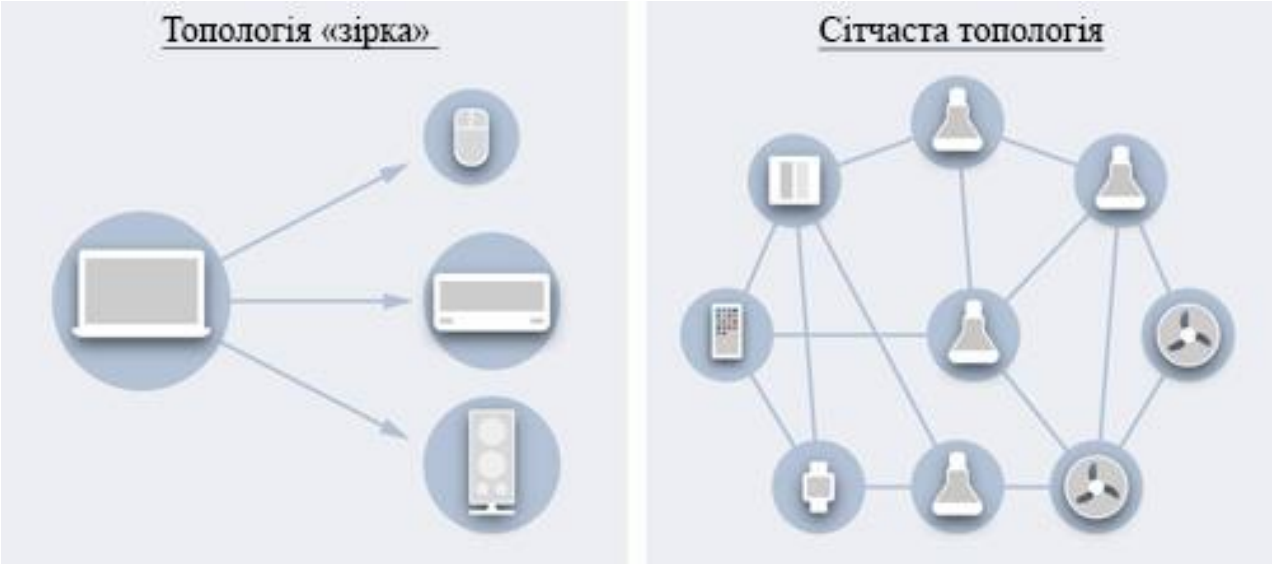


Рисунок 1.5 - Мережа типу "зірка" (зліва), сітчаста мережа (праворуч).

У топології типу «зірка» прийнято розрізняти два підтипи:

1. зірка з пасивним центром;

2. зірка з інтелектуальним центром.

Робота топології типу «зірка» вимагає наявність спеціального багато портового пристрою – концентратора. Окрема лінія передачі з’єднує концентратор з кожним учасником мережі. Це є перевагою, адже , якщо

відмовить одна лінія то доступ до мережі буде втрачено тільки одним учасником [1]. Але недоліком є те, що при виході з ладу концентратора подальша робота в мережі стає неможливою.

Концентратори поділяються на два типи :активні і пасивні. Робота активних концентраторів схожа на роботу репітерів. Вони регенерують і передають сигнали. Оскільки вони мають від 8 до 12 портів для з'єднання з комп'ютером то іноді їх називають багато портовими репітерами [1].

Пасивні концентратори це, наприклад, монтажні панелі чи комутуючі блоки. Вони здатні пропускати через себе сигнал виконуючи роль вузлів комутації при цьому не посилюючи і не відновлюючи його. Пасивні концентратори не потребують підключення до джерела живлення.

Існує тип концентраторів які здатні працювати з кабелями різних типів. Вони називаються гібридними концентраторами. Мережі, що побудовані на концентраторах легко піддаються розширенню завдяки можливості підключення нових концентраторів.

Переваги:

- Дуже легко встановити і управляти мережею топології «зірка», оскільки це найпростіший з видів трафіку, коли справа стосується функціональності.
- Спрощує пошук і усунення несправностей мережі даного типу, так як всі комп'ютери знаходяться в залежності від центрального комп'ютера, що незмінно означає, що будь-яка проблема яка виникає в мережі стосовно непрацездатного стану окремої робочої точки, може бути відстежена на базовій станції.
- У зоряної топології мережі, пакети даних не проходять свій шлях через різні вузли.
- Крім того, той факт, що пакети даних передаються через три різні точки (комп'ютер - маршрутизатор - комп'ютер) гарантує, що дані будуть надійно захищені.

- Так як вузли не пов'язані один з одним, будь-яка проблема в одному конкретному вузлі не перешкоджає продуктивності інших вузлів в мережі.
- Додавання нової машини або проведення заміни старих PC легко проводитися в цій топології мережі, без зриву роботи всієї мережі.

Недоліки:

- Головна проблема топології мережі типу "зірка" це залежність роботи мережі від центрального, вузлового комп'ютера.
- Продуктивність всієї мережі безпосередньо залежить від продуктивності концентратора. Якщо сервер працює повільно, то це призведе до уповільнення роботи всієї мережі.
- Кількість портів в центральному концентраторі обмежує кінцеве число робочих станцій.
- Один з найважливіших недоліків всіх топологій типу зірка є значна витрата кабелю. Це видно на простому прикладі. Якщо, комп’ютери будуть розташовані в одну лінію то при виборі топології зірка знадобиться в кілька разів більше кабелю, ніж при топології шина. В такій ситуації організація мережі топологією зірка буде ускладнювати прокладку кабелю і буде дуже не вигідним економічним рішенням.

Сітчасті-мережі (англ. Mesh-network) - мережева топологія комп'ютерної мережі в якій кожна робоча станція мережі з'єднується з декількома іншими робочими станціями цієї ж мережі з можливим прийняттям на себе функцій комутатора для інших робочих станцій [8]. В Mesh-мережі неможливо контролювати трафік і вузли, що обслуговують мережу, тому що там немає єдиного центру для отримання IP-адрес (DHCP), всі маршрути розподілені і динамічні, і DNS також може бути децентралізовано [8].

Mesh-мережа опціонально анонімна і завжди приватна. Весь трафік шифрується за умовчанням. Немає централізованих логів сесій користувачів і

активності вузлів. Трафік неможливо пріоритезувати. Network Neutrality - це закон, який прописаний в коді.

Мережу неможливо заблокувати або закрити, тому що вона з'єднується за принципом «кожен з кожним», що створює велику кількість зв'язків. Обрив одного або декількох з'єднань не порушить функціонування мережі в цілому.

Якщо сталося стихійне лихо, то за допомогою Mesh-мережі можна швидко побудувати мережу на місці події для зв'язку, а за підтримки ззовні - з'єднати її з глобальною мережею.

В Mesh-мережах точки доступу виконують функції не тільки з надання абонентського доступу , а й функції маршрутизаторів / ретрансляторів для інших точок доступу тієї ж мережі. Завдяки цьому з'являється можливість створення мережі здатної до самовідновлення і самоорганізації.

Архітектура Mesh-мережі представляє собою сукупність кластерів. Вся територія покриття теоретично необмежена і розподіляється на кластерні зони. Кількість точок доступу в одному кластері може бути від 8 до 16 і кожен кластер обов’язково містить одну вузлову точку. Вузлова точка підключатиметься до магістрального інформаційного каналу за допомогою кабелю (оптичного або електричного) або по радіоканалу (з використанням систем широкосмугового доступу).

Всі точки доступу в кластері мають з’єднання між собою транспортному радіоканалу. Точки доступу можуть брати на себе різні функції в залежності від архітектури побудови мережі, наприклад вони можуть виконувати функції ретранслятора (транспортний канал) або функції ретранслятора і абонентської точки доступу.

Для розширення мережі в межах кластера потрібно тільки встановити нові точки доступу. Інтеграція цих точок у вже існуючу мережу відбудеться автоматично.

Переваги і недоліки сітчастої мережі розглянемо на прикладі мережі ZigBee. ZigBee - це відкритий стандарт бездротового зв'язку для систем збору

даних і управління [4]. Технологія ZigBee знайде своє застосування в таких сферах :

- автоматизація домашніх девайсів, датчиків, освітлення, температурного контролю;
- робота з електричними лічильниками і іншими пристроями, що споживають мало енергії (датчики води, електроенергії, датчики задимлення і пожежі);
- периферія персонального комп'ютера;
- інші домашні мережі, що працюють на низьких швидкостях.

Технологія ZigBee споживає набагато менше енергії, працює на низьких швидкостях і буде найкращим рішенням там де потрібно організувати просту мережу, для якої не потрібно використовувати потужні специфікації [5]. Дуже низьке енергоспоживання Zigbee-модулів досягається за рахунок спеціального режиму "сну", коли пристрій не використовується, і низької пропускної здатності передачі даних (250 Кбіт / с проти 300-1000 Мбіт / с у Wi-Fi). Висока пропускна здатність не потрібна ZigBee, адже потрібно передавати зовсім невеликі обсяги даних від датчиків і електроприладів, а ось Wi-Fi повинен забезпечувати перегляд відео в реальному часі і т.п. В результаті Zigbee-модуль може працювати кілька місяців на простій батарейці.

До появи Zigbee в системах розумного будинку намагалися використовувати Bluetooth [2]. Радіус дії у цих протоколів приблизно однаковий, але у Bluetooth занадто великі затримки. Затримка при передачі сигналу у Bluetooth-модуля становить кілька секунд проти 30 мілісекунд у Zigbee-модуля 30мс це набагато менше швидкості реакції у людини - для нас вмикання і вимикання Zigbee -лампочки відбувається моментально.

Технологія Zigbee має переваги і за кількістю вузлів. Нариклад Wi-Fi просто не пристосований до підключення великої кількості учасників до мережі. В теорії до одного Wi-Fi роутера можливо підключити кілька сотень пристроїв, але кожному активному користувачеві відомо, що це не можна досягти на практиці. Роутери середньої цінової категорії починають зависати

і вимагати перезавантаження вже після підключення 15-20 гаджетів: ноутбуків, смартфонів, планшетів та інших девайсів вашої родини. А ось одна Zigbee-мережа може мати тисячі вузлів і при цьому стабільно працювати. Здатність надійно функціонувати при великій кількості підключених пристроїв - необхідна характеристика мережі для розумного будинку і інтернету речей. І звісно не можна забувати про вартість, адже пристрої для розумного будинку з Zigbee-підключенням теж коштують значно дешевше, ніж аналогічні прилади з Wi-Fi [4].

В середині приміщення відстань між робочими станціями мережі становить десятки метрів, а за рахунок ретрансляції цю зону можна розширити до кількох тисяч квадратних метрів.

Більш того, мережа ZigBee в будь-який момент може бути розширена додаванням нових елементів або навпаки розбита на кілька зон простим призначенням відповідного числа нових конфігураторів мережі [5]. Це буває корисно для зниження навантаження і відповідно підвищення швидкості передачі даних.

Для вирішення завдань забезпечення безпеки і автоматизації будинку протоколи, що працюють на частотах нижче 1 ГГц, мають суттєві переваги над більш потужними і функціональними протоколами, які працюють в діапазоні 2,4 ГГц (такими як Wi-Fi, Bluetooth ™ і ZigBee), т.к . для цих завдань потрібна низька швидкість передачі даних.

Такі низькочастотні мережі забезпечують високу дальність зв'язку. Вони можуть працювати на відстанях до кілометра або більше. Ці мережі передають дані безпосередньо одержувачу без посередників на шляху. Однак в цьому випадку на зв'язок сильний вплив надають радіоперешкоди від інших пристроїв. Якщо дані частоти сильно забруднені, то це може накладати істотні обмеження на роботу мережі. Важливою перевагою низькочастотних мереж є більш низьке енергоспоживання в порівнянні з протоколами, які працюють на частоті 2.4 ГГц.

Таблиця 1.1- Порівняльна таблиця характеристик протоколів зв'язку

Технологія	Wi-Fi	Bluetooth	ZigBee
Стандарт зв’язку	IEEE 802.11	IEEE 802.15.4	IEEE 802.15.4
Швидкість передачі даних	300+ Мбіт/с	до 3 Мбіт/с	250 Кбіт/с
Енергоспоживання	Високе	Низьке	Низьке
Частотний діапазон	2.4 Ггц	2.4 Ггц	2.4 Ггц
Підтримка IP	+	-	-
Топологія	"зірка"	"зірка"	"mesh"

1.3 Аналіз існуючих методів зв’язку живлення та обміну інформацією в дротовій схемі підключення

1.3.1 Організація передачі даних по силовим мережам на мікросхемі AMIS-30585.

Компанія ON Semiconductor пропонує для організації передачі даних по силовим мережам спеціалізовану мікросхему модему AMIS-30585.

Для передачі даних в AMIS-30585 використовується S-FSK модуляція (розкид частот по замовчуванню - 10 кГц), а несуча частота програмується в діапазоні 9 ... 95 кГц. Максимальна швидкість передачі 1200 біт / с.

Особливістю даної мікросхеми є наявність вбудованого мікроконтролера з ядром ARM7-TDMI, що забезпечує внутрішню шину реалізації MAC-рівня. Ця особливість є основною перевагою перед рішеннями

інших виробників. Окрім даного модема, ONS пропонує носій-контактний-сумісний модем AMIS-49587 зі швидкістю передачі даних до 2400 біт / с.

Вхідна частина модема на основі рішень від ONS (рис. 1.6) включає в себе: ізолюючий трансформатор з по суті пасивним ФВЧ, драйвер (підсилювач потужності сигналу), приймальний канал, ізолятор на оптроні для отримання синхронізуючого сигналу частотою 50 Гц та додатковий канал для отримання сигналу керування потужністю передавача (зворотній зв'язок з передатчиком).

Передача даних від лічильників за допомогою PLC-модемів найбільш поширена у Франції. З цієї причини виробництвом ізолюючих трансформаторів для таких модемів займаються французькі і німецькі компанії, а самі трансформатори не надто доступні.

З найбільш доступних ринкових варіантів були обрані трансформатори фірми Vigortronix - VTX-111-010 і VTX-111-004

Драйвер лінії реалізований на операційному підсилювачі OPA561 с високим значенням вихідного струму (до ± 1,2 А). Це пов'язано з тим, що драйверу доводиться працювати на навантаження близько 5 Ом. Вихідний струм OPA561 в цій схемі обмежений 0,6 А за допомогою резистора 10 кОм між четвертим виведенням і мінусом живлення. Крім функції розгойдування лінії, OPA561 також виконує функцію ФНЧ. Змодельована АЧХ такого фільтра зображена на рис. 1.7.

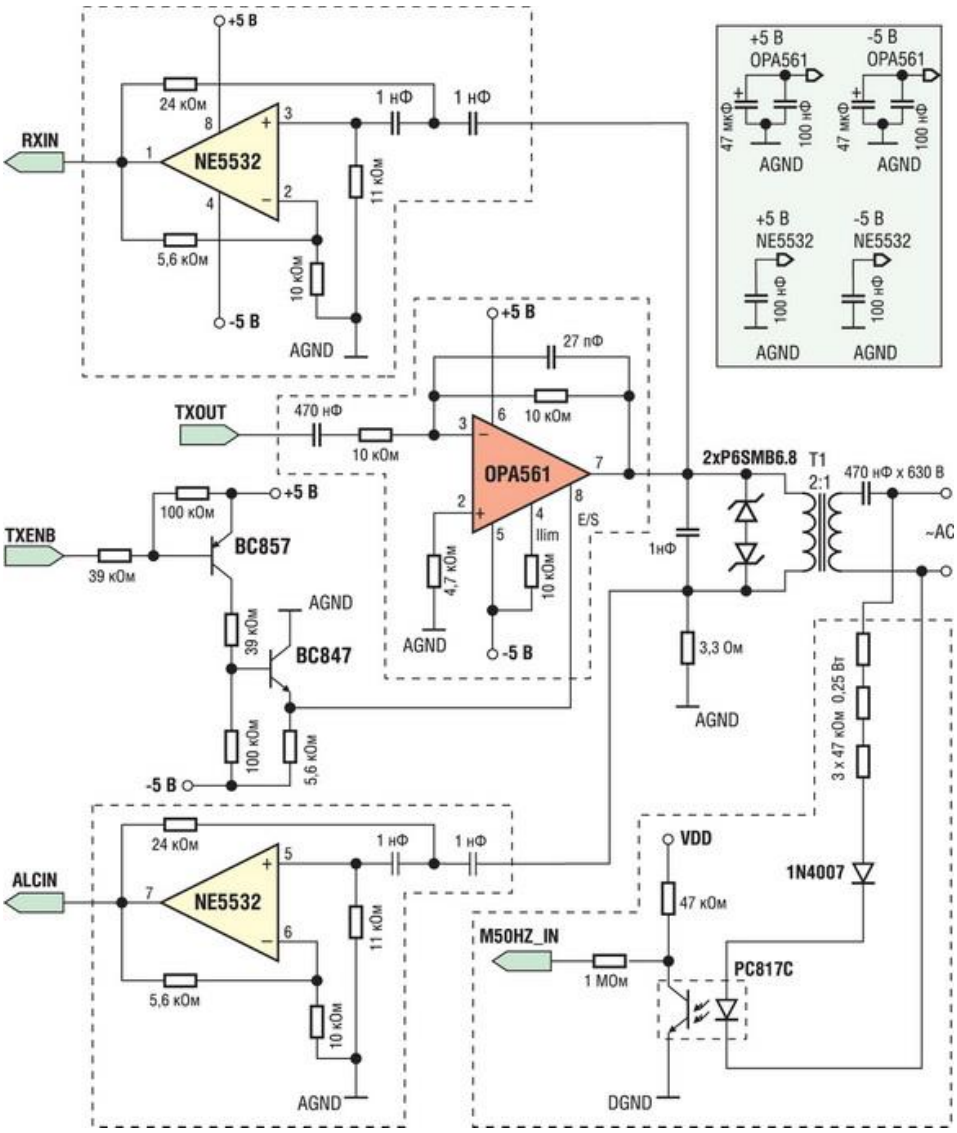


Рисунок 1.6 - Аналогова частина приймально-передаючої схеми

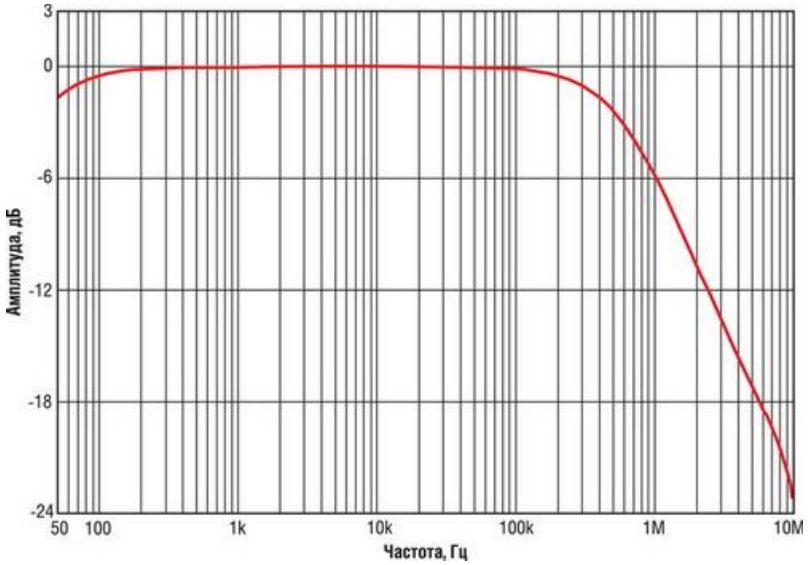


Рисунок 1.7- АЧХ передавача

Оскільки вихід ОУ відключається наявністю на виведенні E / S негативної напруги живлення, а вихідна логіка AMIS-30585 має рівні 0 і +3,3 В, для управління відключенням виходу підсилювача додана схема на транзисторах BC857 і BC847. Слід мати на увазі, що корпус цього ОУ має «Power Pad» для відводу тепла, який слід електрично з'єднати з мінусом живлення.

Приймач і канал управління потужністю передавача схемотехнічно повторюють один одного і реалізовані на здвоєному ОУ NE5532. По суті це - ФВЧ, основне завдання якого - придушити сигнал частотою 50 Гц. Такий фільтр дозволяє отримати послаблення до -90 дБ на частоті 50 Гц. Вихід приймального каналу з'єднується з входом інтегрованого в AMIS-30585 операційного підсилювача, на якому також реалізується ФВЧ з ослабленням порядку -80 дБ, що в сумі дає ослаблення до -170 дБ на частоті 50 Гц. АЧХ фільтра на NE5532 приведена на рис.1.8. Зрозуміло, з урахуванням пасивної фільтрації і вхідні, і вихідна частини є більш вузькосмуговими.

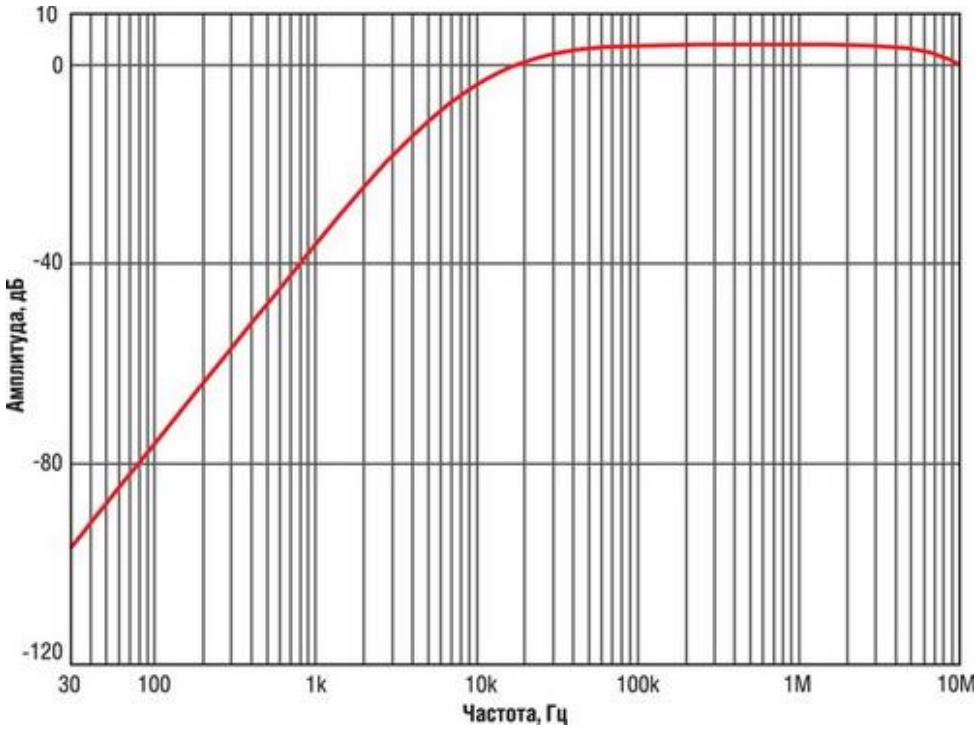


Рисунок 1.8 - АЧХ приймача

Для пакетної передачі даних AMIS-30585 потрібно синхронізуючий сигнал, який несе в собі інформацію про перетин нуля мережевою напругою частотою 50 Гц. Для цієї мети додана схема на оптроні PC817C. Вихідний сигнал цієї

схеми - імпульси частотою 50 Гц, амплітудою від 0 до напруги VDD. Передній і задній фронти цих імпульсів відповідають перетину нуля напруги.

На рис.1.9 зображена спрощена схема включення AMIS-30585. Власне, це мінімум того, що необхідно для роботи даної мікросхеми. В залежності від програми, в якій використовується PLC-модем, розробнику належить вибрати керуючий мікроконтролер.

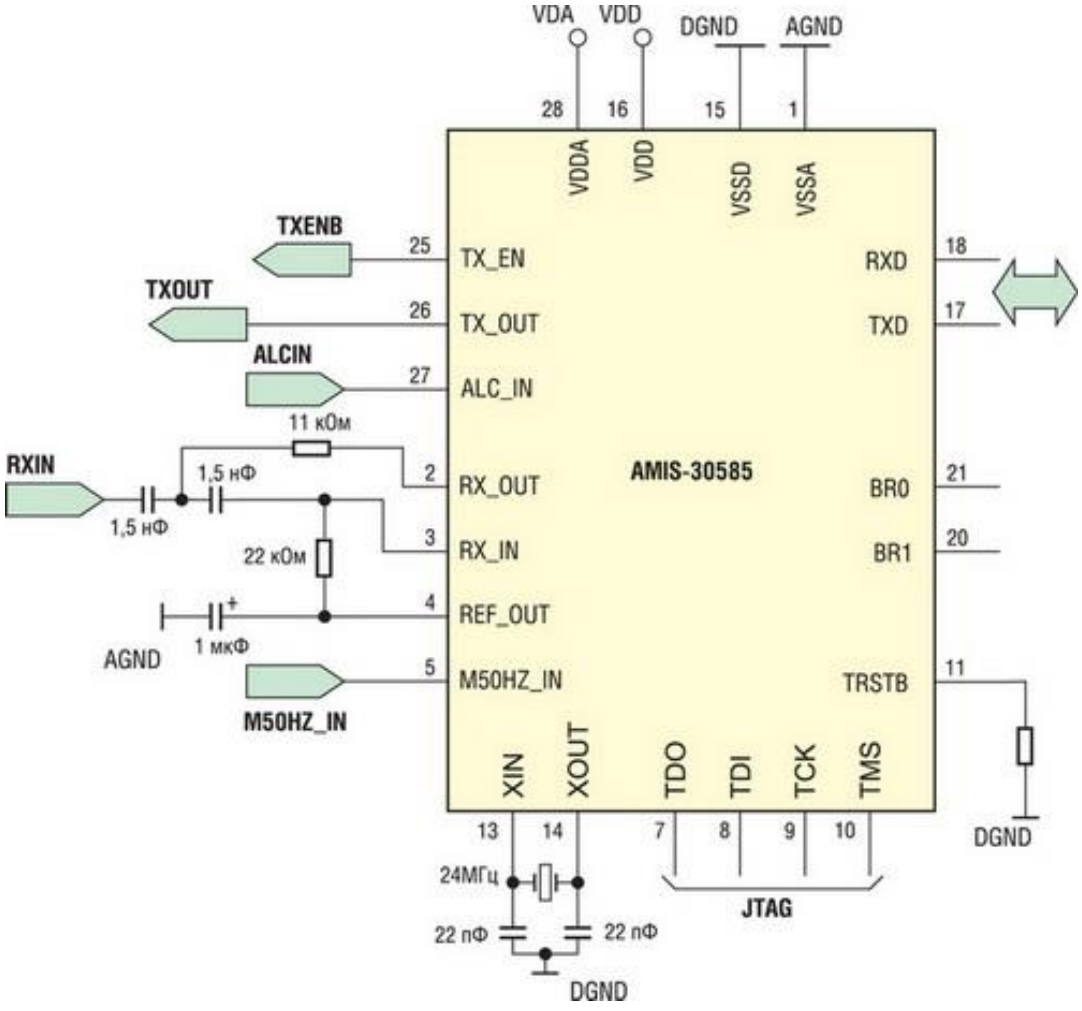


Рисунок 1.9 - Схема підключення AMIS-30585

1.3.2 Рішення для передачі даних з використанням спеціальних мікросхем від STMicroelectronic.

ST7540 - рішення для PLC модему від STMicroelectronics. Відмінною особливістю цієї мікросхеми є наявність інтегрованого підсилювача потужності і двох лінійних стабілізаторів напруги на 5 і 3,3 В. На цьому рішенні можуть зупинитися розробники, які вже мають свій власний протокол передачі даних по послідовному інтерфейсу, наприклад, при переході від

передачі даних по RS-485 до передачі тих же даних за допомогою PLC [3]. Вхідна частина модему на ST7540 показана на рис. 1.10.

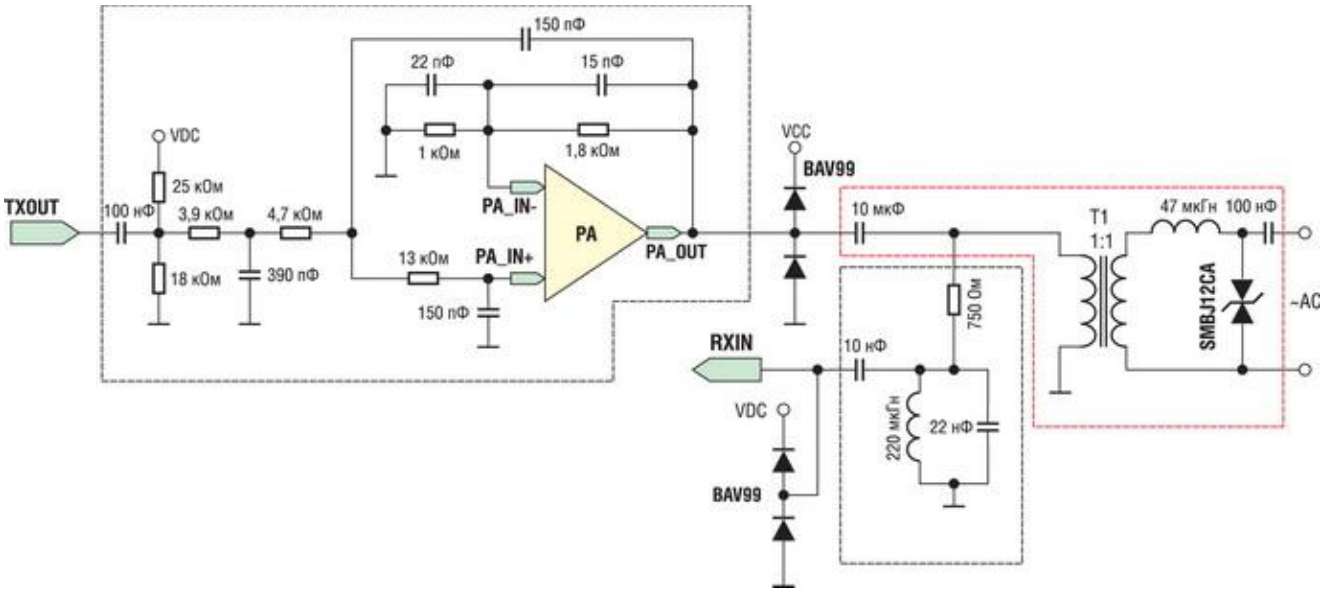


Рисунок1.10 - Аналогова частина приймально-передаючої схеми ST7540

В документації на оцінний набір STMicroelectronics наводить АЧХ приймальної і передавальної частин з урахуванням пасивної частини (виділено червоним на рис. 1.11).

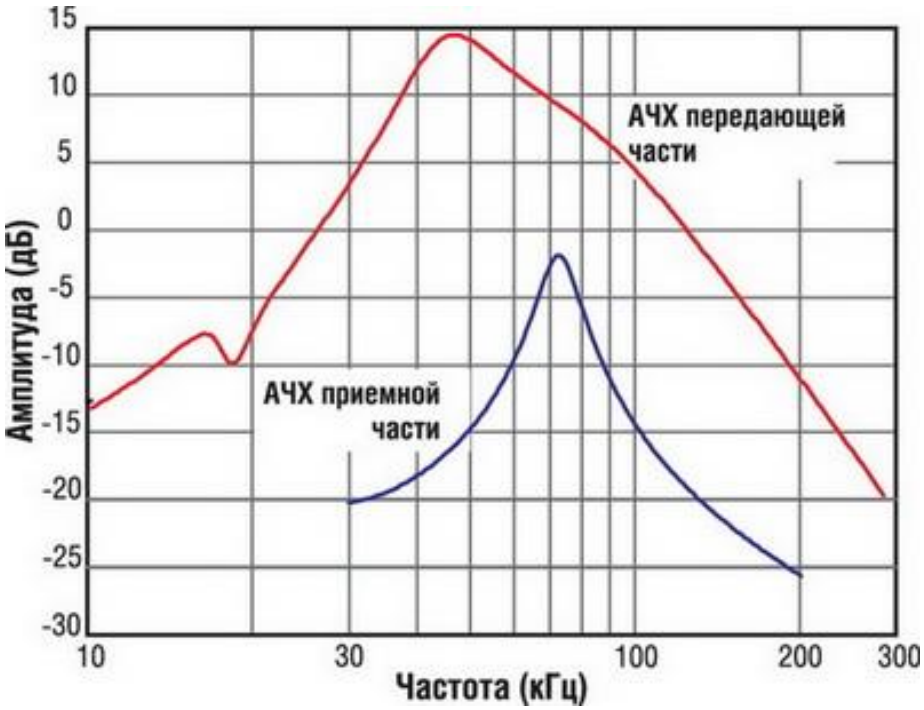


Рисунок 1.11 - АЧХ приймально-передаючої схеми ST7540

1.3.3 Огляд рішення для передачі даних по лінії живлення постійної напруги

Власне розглянуті вище рішення передачі даних по лінії живлення 220В не можуть бути застосовані для передачі даних по довгих лініям постійної напруги без додаткових компонентів.

Запропонована нижче схема (рис. 1.12) реалізує можливість живлення пристроїв на максимальний струм 2А та ведення обміну даними на відстані до 1км.

Схема зв’язку і передачі даних по лінії живлення використовує принцип частотної модуляції і виділенням несущої частоти. Для того, щоб мати можливість створювати частотну модуляцію поверх постійної напруги необхідно щоб лінія мала деякий імпеданс на необхідних нам частотах. Для створення імпедансу використовується індуктивний елемент L1, що розв’язує постійну напругу джерела живлення від напруги модуляції. Такі ж індуктивності використовуються в кожному приймально-передаючому пристрої як фільтр нижніх частот для унеможливлення створення навантаженням імпульсних реактивних струмів що можуть призвести до зменшення імпедансу лінії для високих частот.

Приймаюча частина являє собою пасивний фільтр верхніх частот першого порядку та амплітудний детектор і побудований на элементах C1, R4, C5, VD3, VD2 (рис. 1.13).

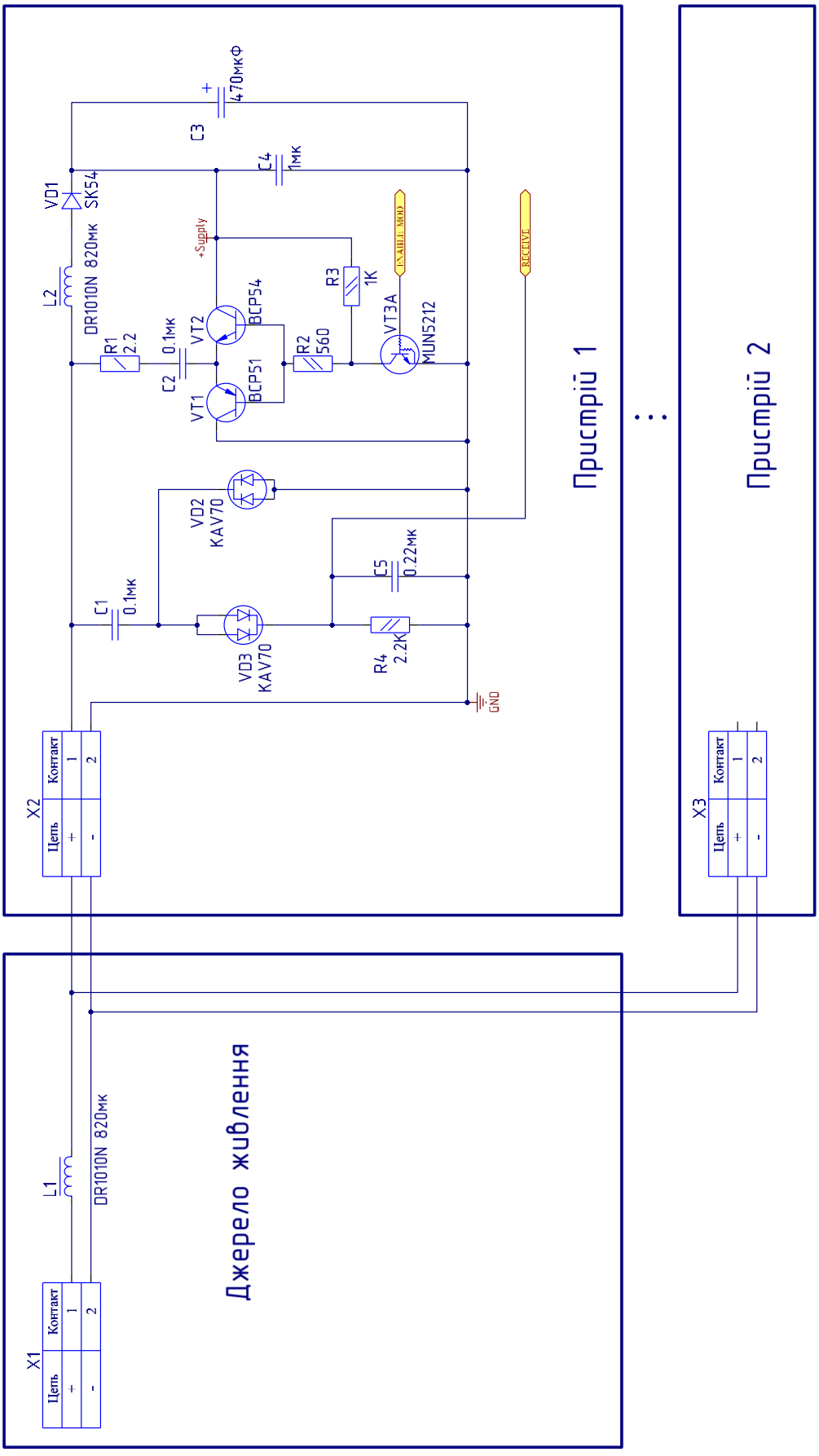


Рисунок 1.12 - Схема реалізації передачі даних на відстані до 1км

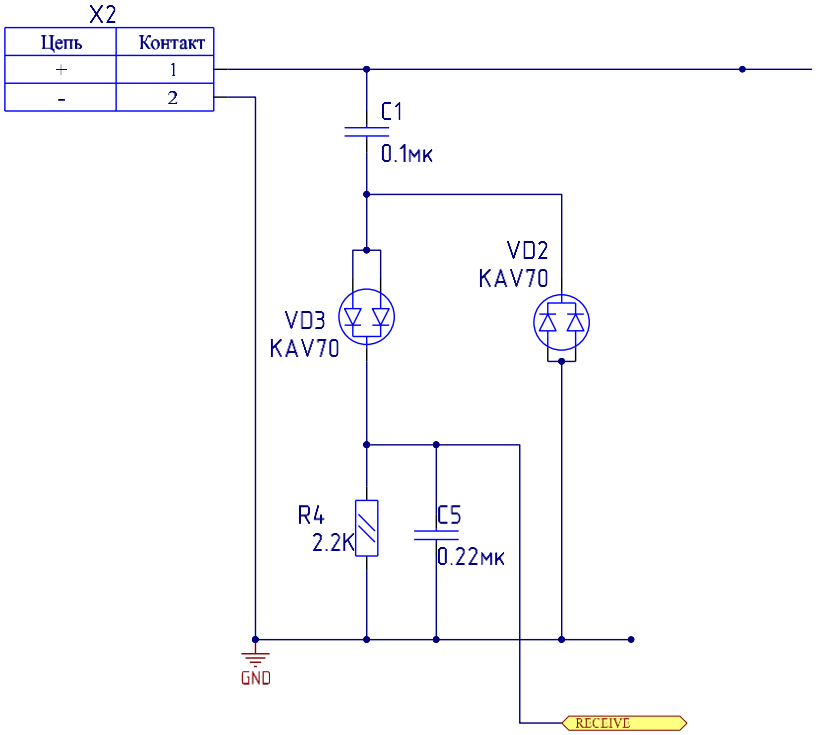


Рисунок 1.13 - Приймальна частина

Приймальна частина відокремлює модульований сигнал і перетворює його в постійну напругу, яка поступає на цифровий вхід мікроконтролера.

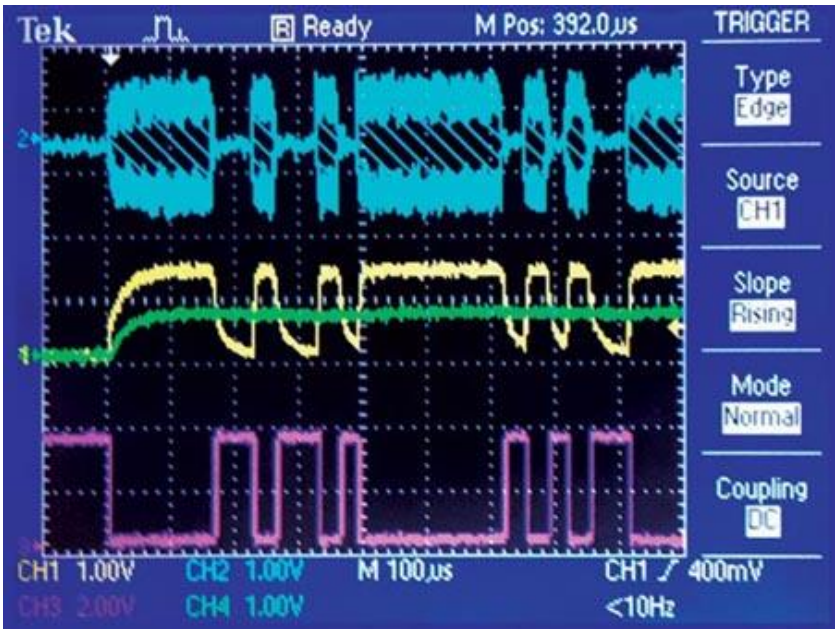


Рисунок 1.14 - Осцилограма відтворення даних

Осцилограма, що ілюструє процес демодуляції і відновлення даних. На цьому малюнку: модульований сигнал (синій), інвертується вхід компаратора (жовтий), відновлені дані (рожевий).

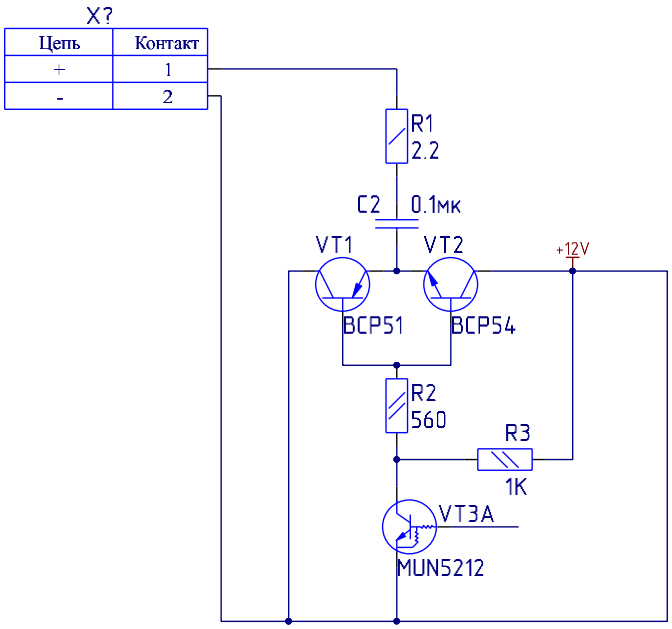


Рисунок1.15 - Передаюча частина

Передаюча частина побудована по принципу PUSH-PULL . На транзистор VT3 подаються імпульси несучої частити. Модуляція здійснюється шляхом високочастотного перемикання конденсатора 2. Резистор R1 зменшує імпульсні струми для зменшення рівня електромагнітних завад.

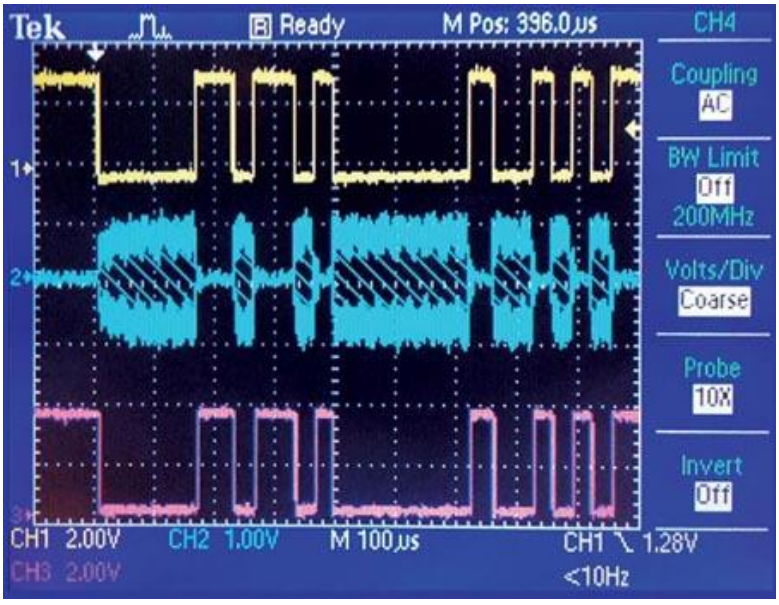


Рисунок 1.16 - Осцилограма обміну даними

Вихідні дані (жовтий) модулюють несучу, передаються по лінії живлення (синій) і точно відновлюються на виході (рожевий).

Багаторівневі системи контролю - це сфера в якій контроль і достовірність інформації є головною вимогою безперебійної роботи системи. При виборі

технологій управління розумним будинком доведеться віддати перевагу провідному, бездротовому або змішаному з'єднанню пристроїв. Вибір однієї з архітектур залежить від конкретної ситуації і переваг людини. Бездротовий спосіб більш гнучкий, власник квартири може розмістити бездротової вимикач (з автономним живленням від батареї) для лампочки в одному місці, а через місяць перенести вимикач в інше місце. Це актуально для людей, що знімають квартиру, так як всю систему можна перенести в інше місце набагато менш болісно, ніж дротову систему. Але в бездротовій системі виникає проблема безпеки, що особливо актуально в багатоповерхових будинках. Бездротову мережу можна прослухати. Наприклад, людина, що проживає поруч, може також мати схожу бездротову систему, що працює на тому ж діапазоні частот і вимикати пристрої в двох сусідніх квартирах одночасно. З провідними системами такої проблеми не виникає.

Також використання передачі даних по лінії живлення є найкращим варіантом для зв’язку пристроїв між приміщеннями з радіочастотним екрануванням або в приміщенням зі щільним металевим покриттям.

2. ОПИС ЗАСОБІВ РОЗРОБКИ

2.1 Огляд студій та бібліотек для роботи з мікроконтролерами серії STM32.

Для розробки ПЗ мікроконтролера необхідне середовище розробки і С-інструментарій. Мікроконтролери на основі ARM-ядер набули великої популярності, а отже, кількість різноманітних варіантів студій розробки - платних і безкоштовних - просто величезна.

Найбільш популярними системами для розробки ПЗ під ARM архітектуру є інструментарії від компаній Keil, IAR Systems, Coocox та Atollic. Це обумовлено найбільш просунутими С-інструментами з точки зору оптимізації і компактності коду. Також, крім лідируючих позицій в С-інструментаріях, дані компанії надають широкі набори додаткового ПЗ - операційні системи реального часу, USB-стеки, TCP / IP-стеки і багато іншого. Слід більш детально розглянути кожну з них.

IAR Embedded Workbench – це високо інтегроване програмне середовище, призначене для розробки коду для мікроконтролерів.

Основні переваги пакету - дружній інтерфейс користувача і неперевершена оптимізація згенерованого коду. Крім цього реалізована підтримка різних операційних систем реального часу і JTAG -адаптер сторонніх компаній.

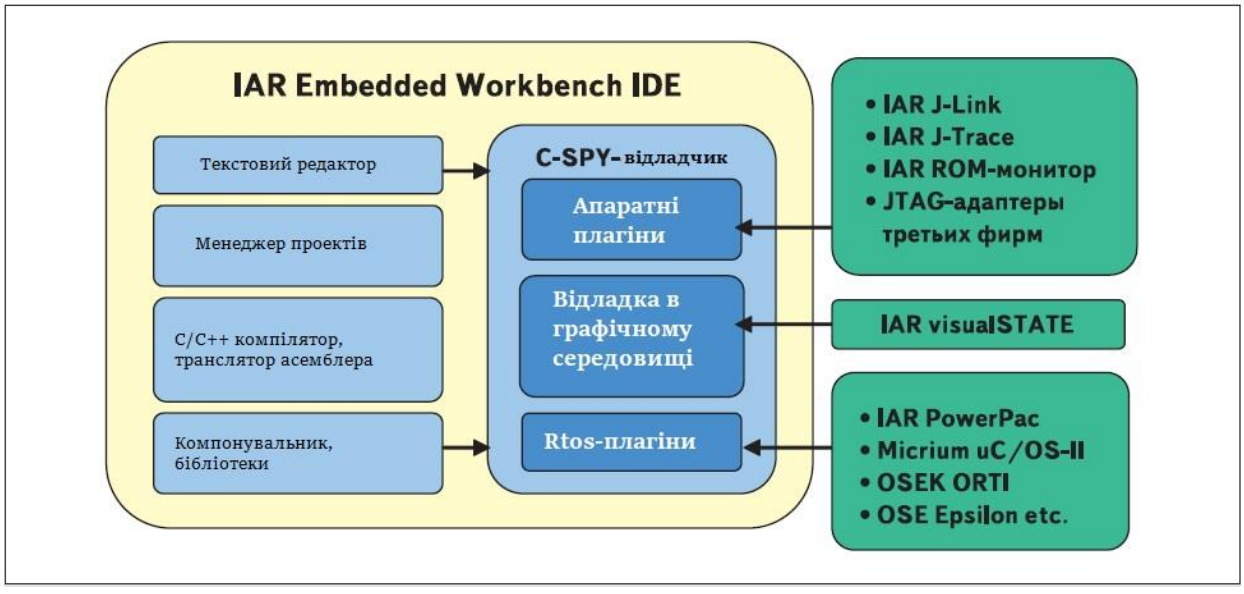


Рис.2.1- Архітектура IAR Embedded Workbench

IAR Embedded Workbench в даний час підтримує роботу з 8-, 16-, 32-розрядними мікроконтролерами.

Програмне середовище включає в себе:

1. C / C ++ компілятор - один з найефективніших в своєму роді. У ньому також присутня повна підтримка ANSI C.

2. Транслятор асемблера, що включає в себе макроассемблер для програм реального часу і препроцесор для C / C ++ компілятора.

3. Компонувальник, що підтримує більше тридцяти різних вихідних форматів для спільного використання з внутрішньосхемними емуляторами.

4. Текстовий редактор, налаштований на синтаксис мови C,що має зручний для користувача інтерфейс, автоматичне виділення помилок в програмному коді, можливість налаштування інструментальної панелі, підсвічування блоків, а також зручну навігація по іменах підпрограми, макроси і змінні.

5. Симулятор і відладчик в кодах C і асемблера дають змогу перегляди регістри введення/виведення, області пам’яті EEPROM , DATA, CODE, а також встановлювати точки зупинки, прапори. Відладчик дає змогу контролювати стек і відтворювати покрокове виконувати програму. Якщо відладчик відсутній, то на допомогу приходить симулятор, який, однак, не має можливості емулювати роботу процесора.

6. Менеджер проектів дає змогу управляти робочими модулями і контролювати їх.

7. Додаткові утиліти для роботи з оптимізованою CLIB / DLIB бібліотекою.

Ще одна, набираючи популярність студія розробки це Atollic TrueSTUDIO. Середовище розробки Atollic TrueSTUDIO було створене на базі популярної платформи з відкритим вихідним кодом - Eclipse. Програма включає в себе: редактор коду (з підтримкою мов C / C ++ і асемблера), менеджер проектів, C / C ++ компілятор і систему збирання вихідного коду (з можливістю паралельної компіляції), відладчик (з підтримкою функції

багатоядерного налагодження), а також різні додаткові інструменти для: аналізу коду (відповідно до стандарту кодування MISRA-C), управління завданнями, відстеження змін / оновлень, пристрій автоматичного тестування створеного коду.

Atollic TrueSTUDIO підтримує безліч ARM-ядер (сімейств ARM9, ARM7, Cortex-A, Cortex-R, Cortex-M і т.д.) від таких відомих вендорів, як Atmel, Infineon, Freescale, NXP, Silicon Labs, Renesas, Spansion , STMicroelectronics, Toshiba, Texas Instruments і деяких інших. Також є підтримка двоядерних і багатопроцесорних пристроїв. Офіційно середовище розробки взаємодіє з відладчиками ST-LINK (і ST-LINK / V2) від STMicroelectronics, SAM-ICE від Atmel, Segger J-Link, OSJTAG і P & E Multilink.

Atollic TrueSTUDIO включає в себе професійний відладчик C / C ++ (на базі відладчика GNU gdb) . Atollic TrueSTUDIO пропонує відгадчики для:

- P & E Micro
- SEGGER J-Link / J-Trace
- ST-Link
- OpenOCD (не в комплекті, корисний для mbed пристроїв)

Відладчик здатний працювати з одноядерними, багатоядерними та багатопроцесорними системи, а також відлагоджувати як вбудований код мікроконтролера, так і програми командного рядка Windows PC.

Налагоджувач містить багато потужних функцій, зокрема:

- Простий і розширений контроль виконання потоку налагодження
- Дисплей стека викликів
- Прості / розширені візуалізаційні змінні / об'єкти
- Прості / складні точки зупинки та точки спостереження
- Відображення основних регістрів процесора
- Серійний / Telnet / SSH термінал
- Розширене відображення периферійних регістрів
- Режим перегляду змінної в реальному часі

- Системний аналіз і трасування в реальному часі (SWV)
- Підтримка трасування з використанням ETM / ETB / MTB, включаючи тригери початку / зупинки сліду
- Інструмент аналізу несправностей ЦП
- Kernel-aware debug support для багатьох RTOSes

Відладчик має інформацію про RTOS, і має функції для відстеження інструкцій, а також системного аналізу та відстеження подій та даних у режимі реального часу, а також включає потужну мову сценаріїв для автоматизації розширених завдань відлагодження. Інструмент аналізу несправностей ЦП є зручною функцією для розуміння умов несправності ЦП.

Atollic надає спеціальну підтримку ядра, щоб спростити налагодження для наступних RTOSes:

- embOS
- eTaskSync
- FreeRTOS
- MQX
- OpenRTOS
- RTXC
- ThreadX
- TOPPERS (uITRON)
- uC/OS-III

Дана IDE має власний майстер налаштувань, що допомагає прописати всі ключові параметри додатку: назва і тип проекту, виробника чіпа, сімейство і модель мікроконтролера, модель відладочної плати, місце розміщення коду в пам'яті пристрою і багато іншого. Виконуваний файл (main.c), представлений в інспекторі коду, за замовчуванням формується з шаблону, підготовленого для декількох плат. Він містить досить багато зайвої інформації, яку при бажанні можна видалити. Вбудована система налагодження нічим не відрізняється від аналогічних програм, присутні всі стандартні інструменти.

Під час роботи процесу налагодження на екран виводиться різноманітна інформація - фрагменти виконуваного в даний момент коду, фрагменти коду асемблера, що відповідають виконуваній інструкції, значення поточних змінних. Середовище розробки Atollic TrueSTUDIO генерує файли прошивки унікального формату - * .elf. Загалом Atollic TrueSTUDIO не потребує довгих налаштувань, щоб почати роботу але з недоліків слід відмітити, що інтерфейс користувача не є комфортним і інтуїтивно зрозумілим. Після того, як STMicroelectronics викупив Atollic TrueSTUDIO ця студія розробки стала повністю безкоштовною.

Coocox IDE - це багатофункціональне середовище розробки, яке обладнане ефективним C / C ++ компілятором. Функціонал середовища CooCox CoIDE дозволяє завантажувати вихідний код програми, редагувати його, проводити компіляцію (сторонніми засобами), прошивати контролер і проводити налагодження. Програма заснована на базі Eclipse і має всі її переваги. Coocox IDE містить зручний редактор коду, що підсвічує синтаксис і додає підказки при написанні коду. Присутні функції глобальної заміни змінної та пропозиції варіантів закінчення коду. Середовище підтримує мікроконтролери серії ST, а також ряд інших сімейств: Atmel, Holtek, Freescale, Nuvoton, NXP, Energy Micro, Texas Instruments і деякі інші. З кожним оновленням CoIDE підтримує все більшу кількість архітектур. Під час створення нового проекту пропонується вибір бібліотек і мікросхеми, а також надаються короткі характеристики на обраний контролер. До складу студії входить відладчик ST-Link, який дає змогу контролювати вміст регістрів і забезпечує покрокове виконувати програми.

В функціонал CoIDE входить створення всієї структури проекту, а також підключення бібліотек з готовими прикладами.

Перевагою даного середовища є те, що воно повністю безкоштовне і для завантаження потрібно лише пройти процедуру реєстрації. Також розроблено спеціальний менджер CoCenter через який можна завантажити середовище розробки і різні додаткові утиліти розробника.

У списку CoCenter знаходяться:

- CooCox CoIDE-студія розробки;
- CooCox CoFlash- софт для програмування Flash-пам'яті;
- CooCox CoSmart- конфігуратор виводів мікроконтролера;
- CooCox CoOS- операційна система реального часу;
- CooCox ColinkEx USB Driver і CooCox MDK Plugin - потрібні для відладчика -программатора CoLink.

CooCox CoIDE є одним з найпростіших і швидких в плані встановлення, освоєння і налаштування рішень, що дозволяє навіть починаючим користувачам досягати в ньому істотних результатів. Зробити правильні налаштування та успішно пройти через всі етапи розробки допомагає вбудований в середовище майстер.

Головним недоліком середовища CoIDE є відсутність компілятора GCC. Компілятор потрібно окремо завантажувати і встановлювати і , щоб розпочати з ним роботу, в налаштуваннях студії потрібно прописати правильний шлях до нього. Це створює певні незручності. Для контролерів ARM існує декілька варіантів компіляторів, а CoIDE розроблялася для взаємодії з ARM GCC.

Також до недоліків потрібно віднести те, що шляхи до файлів проекту жорство прописані в програмі. Якщо проект було переміщено потрібно вручну прописувати шляхи до нього, а інакше зібрати проект буде неможливо. Також професійним розробникам середовище CoIDE може здатися занадто простим і незручним через відсутність можливості проводити тонкі налаштування самої IDE.

Для розробки в рамках даного дипломного проекту я обрала середовище розробки Keil uVision, оскільки, це середовище має більш зручний інтерфейс користувача і нічим не поступається IAR Embedded Workbench. Keil uVision представляє набір утиліт для виконання повного комплексу заходів з написання програмного забезпечення для мікроконтролерів. Головною перевагою цієї системи є те, що вона дозволяє працювати з проектами будь-якого ступеня складності, починаючи з введення і редагування вихідних

текстів і закінчуючи внутрішньосхемним налагодженням коду і програмуванням ПЗУ мікроконтролера. Серед основних програмних засобів Keil uVision можна відзначити:

- базу даних мікроконтролерів, що містить докладну інформацію про всі підтримувані пристрої;
- менеджер проектів,що слугує для об'єднання окремих текстів програмних модулів і файлів в групи, оброблюваних за єдиними правилами;
- вбудований редактор, який полегшує роботу з вихідним текстом за рахунок використання багатовіконного інтерфейсу та виділення синтаксичних елементів шрифтом і кольором;
- засоби автоматичної компіляції, ассемблювання і компонування проекту, які призначені для створення виконуваного (завантажувального) модуля програми;
- симулятор відладчика відлагоджує роботу скомпільованої програми на віртуальній моделі мікропроцесора;
- додаткові утиліти, що полегшують виконання найбільш поширених завдань.

Програмний продукт STM32CubeMX дає змогу швидко налаштувати периферію мікроконтролера. CubeMX генерує код ініціалізації периферії мікроконтролерів сімейства STM32 і тим самим швидко налаштовую всю потрібну користувачеві периферію.

STM32Cube є комплексним програмним рішенням, що комбінує вбудоване програмне забезпечення елементів MCU на базі програмного забезпечення STM32CubeMX. Вбудоване програмне забезпечення не тільки охоплює всі мікроконтролери STM32 з високою переносимістю драйверів низького рівня, але поставляється з набором компонентів middleware рівня, таких як RTOS, USB, TCP / IP, файлова система або графіка. STM32CubeMX допомагає користувачеві налаштувати STM32 MCU (терморегулятори, ланцюги тактування і периферію) і програмне забезпечення стеку. Вона також

може допомогти оцінити енергоспоживання завдяки калькулятору розрахунку споживаної потужності. У STM32Cube вбудовані бібліотеки програмного забезпечення і STM32CubeMX генератор коду можуть бути використаний незалежно один від одного, але їх повний потенціал досягається коли вони використовуються разом; як тільки MCU налаштований, користувач може генерувати код ініціалізації, заснований на виставлених налаштуваннях в STM32CubeMX.

2.2 Огляд сучасних операційних систем реального часу

При написанні програмного забезпечення використовувалися операційні системи реального часу. Операційні системи реального часу (ОСРЧ (RTOS)) призначені для забезпечення інтерфейсу до ресурсів критичних за часом систем реального часу.

Використання ОСРЧ дає багато переваг розробнику, а саме :

- Багатозадачність. ОСРЧ надає програмісту готовий та налагоджений механізм багатозадачності. Кожну окрему задачу можна програмувати окремо, не звертаючи уваги на інші задачі. Програмісту не потрібно піклуватися про перемикання між задачами, замість нього це зробить ОСРЧ відповідно до алгоритму роботи планувальника.
- Обмін даними між задачами. Завдяки використанню черги можна бути впевненим, що дані дійдуть від відправника до адресата.
- Синхронізація. Це дуже важливо коли різні задачі звертаються до одного і того ж апаратного ресурсу. М`ютекси, та критичні секції допоможуть синхронізувати роботу з периферійними пристроями.

Крім того, одна і таж С для МК може виконуватись на великій кількості архітектур мікроконтролерів. Це дає змогу швидко перенести розробку на нову апаратну платформу, коли застаріває мікроконтролер або з'являється нова і більш вигідна модель на ринку. Зміни торкнуться тільки частини коду, що відповідає за звернення до вбудованої периферії.

Використання ОСРЧ призводить до певних витрат. це:

- Додаткова витрата пам'яті програм для зберігання ядра ОСРЧ.
- Додаткова витрата пам'яті даних для зберігання стека кожного завдання, семафорів, черг, м'ютексів та інших об'єктів ядра операційної системи.
- Додаткові витрати часу процесора на перемикання між завданнями.

Розглянемо специфіку роботи з операційною системою та її складові.

Кожна програма, що виконується являє собою задачу (Task). Якщо ОС дозволяє одночасно виконувати безліч завдань, вона є мультизадачною (Multitasking). Більшість процесорів можуть виконувати тільки одну задачу в один момент часу. Однак за допомогою швидкого перемикання між завданнями досягається ефект паралельного виконання всіх завдань. На рис. 1 показано істинно паралельне виконання трьох завдань.

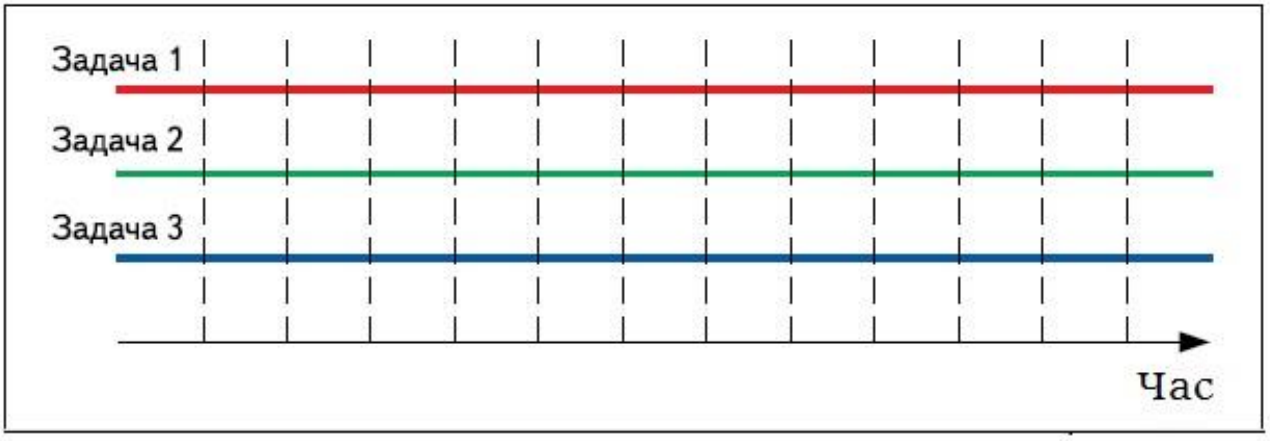


Рисунок 2.2 - Паралельне виконання задач

У реальному ж процесорі при роботі ОСРЧ виконання завдань носить періодичний характер: кожна задача виконується певний час, після чого процесор «перемикається» на наступне завдання (рис. 2).

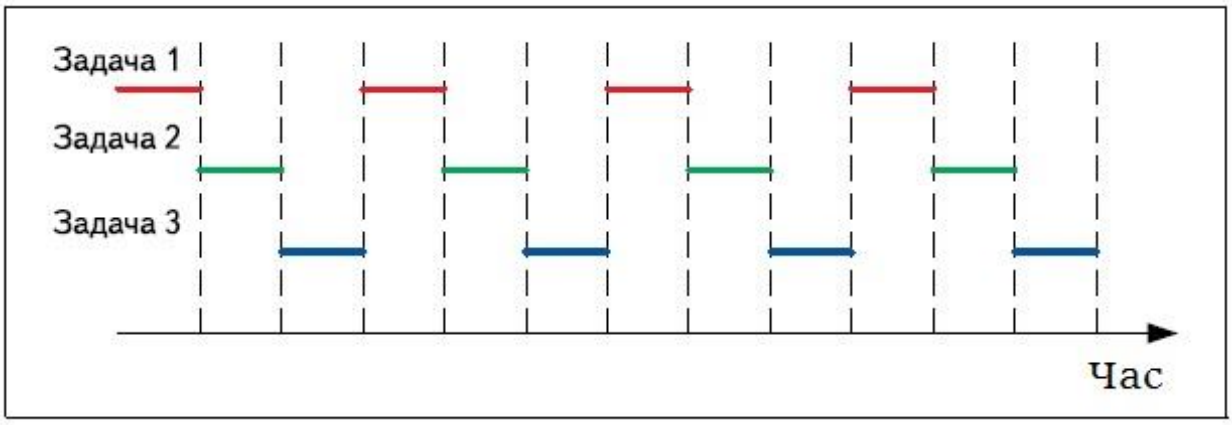


Рисунок 2.3 - Розподілення процесорного часу між кількома задачами ОСРЧ

Планувальник (Scheduler) - це частина ядра ОСРВ, яка визначає, яка з задач, готових до виконання, виконується в даний конкретний момент часу. Планувальник може призупиняти, а потім знову відновлювати виконання задачі протягом усього її життєвого циклу (тобто з моменту створення задачі до моменту її знищення).

Робота планувальника RTOS в режимі витісняючої багатозадачності має багато спільного з алгоритмом перемикання потоків в сучасних ОС загального призначення. Витісняюча багатозадачність передбачає, що будь-яка задача з низьким пріоритетом під час виконання може перерватися готовою до виконання задачею з більш високим пріоритетом. Як тільки задача з високим пріоритетом виконала свої дії, вона завершує свою роботу або переходить в стан очікування, і управління знову отримує задача з низьким пріоритетом. Перемикання між задачами здійснюється через рівні кванти часу роботи планувальника, тобто високо пріоритетна задача, як тільки вона стала готова до виконання, очікує завершення поточного кванта, після чого управління отримує планувальник, який передає управління задачі з високим пріоритетом. Таким чином, час реакції RTOS на зовнішні події в режимі витісняючої багатозадачності - не більш одного кванта часу планувальника, який можна задавати в налаштуваннях. За замовчуванням він дорівнює 1 мс. Якщо готові до виконання кілька завдань з однаковим пріоритетом, то в такому випадку планувальник виділяє кожної з них по одному кванту часу,

після закінчення якого керування отримує наступне завдання з таким же пріоритетом, і так далі по колу. Кооперативна багатозадачність відрізняється від витісняючої тим, що планувальник самостійно не може перервати виконання поточного завдання, навіть якщо з'явилася готова до виконання задача з більш високим пріоритетом. Кожне завдання має самостійно передати управління планувальнику. Таким чином, задача з високим пріоритетом чекатиме, поки задача з низьким завершить свою роботу і передасть управління планувальнику. За таких умов важко прогнозувати час реакції системи на зовнішню подію, оскільки неможливо визначити як довго буде виконуватися поточне завдання і коли відбудеться передача управління.

Витісняюча і кооперативна концепції багатозадачності об'єднуються разом в гібридній багатозадачності, коли виклик планувальника відбувається кожен квант часу, але, на відміну від витісняючої багатозадачності, програміст має можливість зробити це примусово в тілі задачі. Найефективнішою ця концепція багатозадачності є, коли необхідно скоротити час реакції системи на переривання. Припустимо, наразі виконується задача з низьким пріоритетом , а задача з високим очікує настання деякого переривання. Далі відбувається переривання, але після закінчення роботи обробника переривань виконання повертається до поточної фонової задачі, а задача з високим пріоритетом очікує, поки закінчиться поточний квант часу. Однак якщо після виконання обробника переривання передати управління планувальнику, то він передасть управління задачі з високим пріоритетом, що дозволяє значно скоротити час реакції системи на переривання, пов'язане з зовнішньою подією.

Будь-яка програма, яка виконується під управлінням RTOS, являє собою безліч окремих незалежних завдань. Кожне завдання виконується в своєму власному контексті без випадкових залежностей від інших завдань і ядра RTOS. Тільки одна задача з безлічі може виконуватися в один момент часу, і планувальник відповідальний, яка саме. Планувальник зупиняє і відновлює

виконання всіх завдань по черзі, щоб досягти ефекту одночасного виконання декількох завдань на одному процесорі.

Найпростіший спосіб організувати обмін інформацією між завданнями - використовувати загальну глобальну змінну. Доступ до такої змінної здійснюється одночасно з декількох завдань. Однак такий підхід має істотний недолік: при спільному доступі декількох завдань до загальної змінної виникає ситуація, коли виконання однієї задачі переривається планувальником саме в момент модифікації загальної змінної, коли та містить не остаточне (спотворене) значення. При цьому результат роботи іншої задачі, яка отримає управління і звернеться до цієї до цієї змінної, також виявиться спотвореним. Вирішити подібні проблеми дозволяє використання черг для передачі інформації між завданнями. У RTOS черги представляють собою фундаментальний механізм взаємодії завдань один з одним. Вони можуть бути використані для передачі інформації як між завданнями, так і між перериваннями і завданнями. Основна перевага використання черг - це те, що їх використання є безпечним в багатозадачному середовищі (thread safe). Тобто при використанні черг автоматично вирішується проблема спільного доступу декількох задач до одного апаратного ресурсу, роль якого в даному випадку відіграє пам'ять. Черга - це простий FIFO (хоча також можна писати в початок черги, а не в кінець) буфер, який може зберігати фіксоване число елементів, відомого розміру. Запис в чергу - це побайтове копіювання даних в буфер, читання - копіювання даних і видалення з черги. Черга - це самостійний об'єкт ядра, вона не належить жодному конкретному завданню. Навпаки, будь-яка кількість завдань може як читати, так і записувати дані в одну і ту ж чергу. Слід зазначити, що ситуація, коли в чергу поміщають дані відразу кілька завдань, є «звичайною справою» для програм під керуванням ОСРЧ однак читання даних декількома завданнями з однієї черги зустрічається рідко.

Вбудовуванні мікроконтролерні системи функціонують, відповідаючи діями на події зовнішнього світу. Наприклад, отримання Ethernet-пакета (подія) вимагає обробки в задачі, яка реалізує TCP / IP-стек (дія). Зазвичай

вмонтовані системи обслуговують події, які приходять від безлічі джерел, причому кожна подія має свою вимогу за часом реакції системи і витрат часу на його обробку. При розробці вбудовуваної микроконтроллерной системи необхідно підібрати свою стратегію реалізації обслуговування подій зовнішнього світу.

Переривання - це подія (сигнал), що змушує мікроконтролер змінити поточний порядок виконання команд. В момент спрацювання флагу переривання припиняється поточна послідовність виконання команд і управління переходить в обробник переривання. Обробник переривання можна уявити у вигляді функції . Оброблювач переривання реагує на подію і виконує послідовність дій, що прописана в ньому, після чого повертає управління в перерваний код.

Двійкові семафори призначені для ефективної синхронізації виконання завдання з виникненням переривання. Вони дозволяють переводити завдання зі стану блокування в стан готовності до виконання кожен раз, коли відбувається переривання. Це дає можливість перенести більшу частину коду, що відповідає за обробку зовнішнього події, з обробника переривання в тіло завдання, виконання якої синхронізовано з відповідним перериванням. Усередині обробника переривання залишиться лише невеликий, швидко виконується фрагмент коду. Кажуть, що обробка переривання відкладена і безпосередньо виконується завданням-обробником. Типовий механізм роботи двійкового семафору можна побачити на рис.2.4.

На рис. 2.5 видно, що переривання перериває виконання одної задачі і повертає управління іншій. У момент часу (1) виконується прикладна задача, коли відбувається переривання при виникненні якогось зовнішнього події.

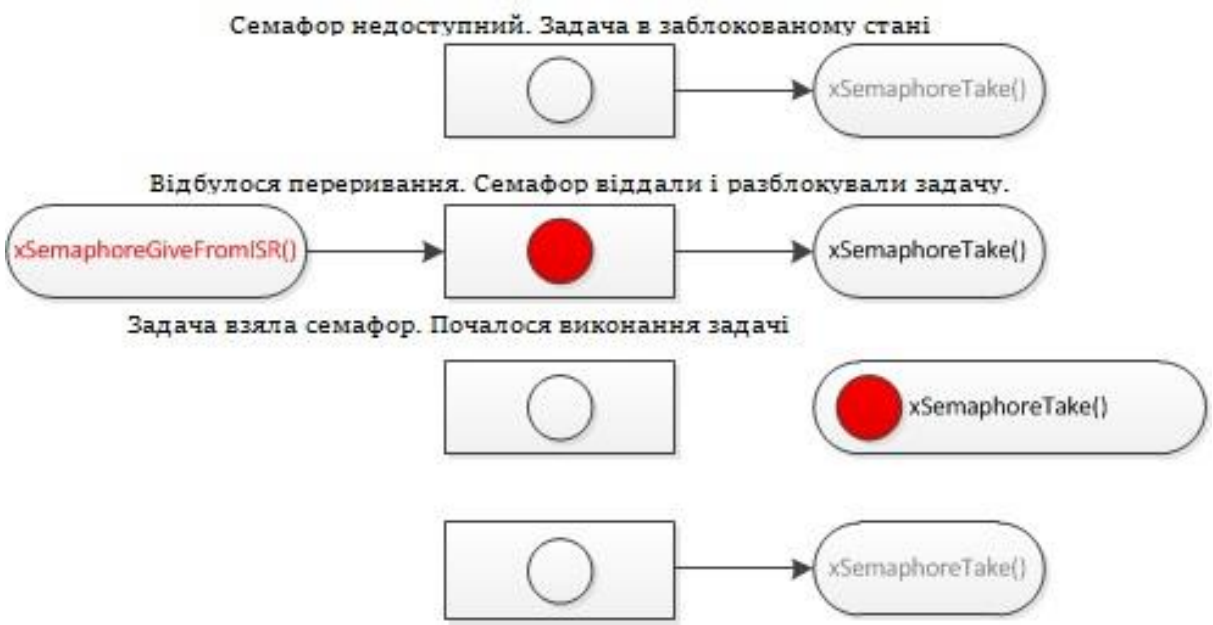


Рисунок 2.4 - Механізм роботи семафору

У момент часу (2) управління отримує обробник переривання, який, використовуючи механізм довічного семафора, виводить з блокованого стану завдання-обробник переривання. Так як пріоритет задачі-обробника вище пріоритету прикладної задачі, то завдання-обробник витісняє прикладну задачу, яка залишається в стані готовності до виконання (3). У момент часу (4) задача-обробник блокується, чекаючи виникнення наступного переривання, і управління знову отримує фонова прикладна задача.

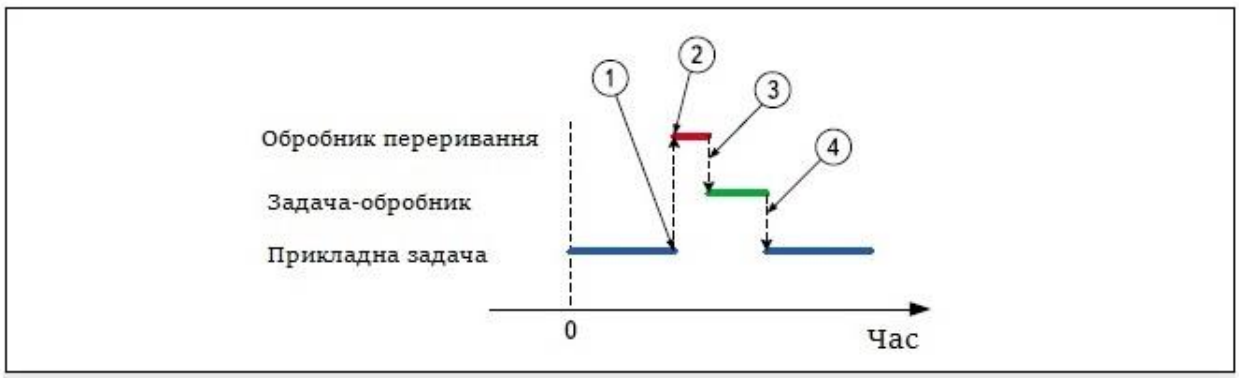


Рисунок 2.5 - Відкладена обробка переривання з використанням двійкового семафору

Коли потрібно реалізувати спільний доступ до ресурсу двох або більше завдань на допомогу приходить м'ютекс. При використанні в якості механізму взаємного виключення м'ютекс можна уявити як семафор, що відноситься до

ресурсу, доступом до якому необхідно управляти. На відміну від семафора м'ютекс у RTOS надає механізм успадкування пріоритетів, про яке буде розказано нижче. Також слід зазначити, що використання мьютекса з тіла обробника переривання неможливо. Щоб коректно отримати доступ до ресурсу, задача повинна попередньо захопити м'ютекс, стати його власником. Коли власник семафора закінчив операції з ресурсом, він повинен віддати м'ютекс назад. Тільки коли м'ютекс звільнився (повернутий будь-якою задачею), інша задача може його захопити і безпечно виконати свої операції із загальним для декількох завдань ресурсом. Задачі не дозволено виконувати операції з ресурсом, якщо в даний момент вона не є власником м’ютекса.

Розглянемо найпопулярніші рішення серед операційних систем реального часу для мікроконтролерів.

Keil RTX це детермінована операційна система реального часу розроблена для процесорів ARM і Cortex-M. Вона дозволяє створювати програми які можуть одночасно виконувати кілька завдань і допомагає створювати додатки які краще структуровані і легко супроводжувані.

Можливості:

- Гнучкий планувальник: циклічна, витісняюча і кооперативна багатозадачність;
- Швидкодіюча робота в реальному часі з мінімальним часом відгуку на переривання;
- Маленький об’єм для систем з обмеженими ресурсами;
- Необмежена кількість задач з 254-ма рівнями пріоритету кожна.
- Необмежена кількість поштових скриньок (mailboxes), семафорів, м'ютексів і таймерів
- Підтримка багатопоточної і потоконезалежної роботи

Перевагами Keil RTX э те, що вона безкоштовна, легко портується на контролери Arm і є різноманітні бібліотеки для роботи з нею. З недоліків потрібно відмітити те, що вона придатна для роботи тільки с контролерами архітектури Arm, має скорочений функціонал , за швидкістю програє іншим

операційним системам реального часу і тому може підійти тільки для малих проектів та для навчання.

MicroC / OS-II - операційна система реального часу с гарною підтримкою витісняючої багатозадачності і низькою вартістю. Повністю написана на мові програмування С, завдяки чому легко переноситься на різні платформи. На даний момент μC / OS-II перенести більш ніж на сто різних платформ, серед яких є і 8-бітові мікроконтролери. Має застосування в авіації і медичному обладнанні.

MicroC / OS-II пропонує наступні переваги :

- Задачі(потоки)
- Флаги подій
- Передача повідомлень
- Менеджер пам’яті
- Семафори
- Менеджери часу

Переваги :

- Портативність. Представляючи небувалу просту у використанні, ядра MicroC / OS постачаються з повним вихідним кодом та глибинною документацією. Ядра MicroC / OS працюють на великій кількості архітектур процесорів.
- Масштабований. Ядра MicroC / OS дозволяють виконувати необмежені завдання та об'єкти ядра. Ступінь пам'яті ядра може бути зменшена, щоб містити лише функції, необхідні для вашої програми, як правило, 6-24 Кбайт коду та 1 Кбайт простору даних.
- Надійний. Ядра MicroC / OS включають функцію налагодження, що зменшує час розробки. Ядра забезпечують широкий діапазон перевірок, включаючи перевірки вказівників, переданих у викликах API, аргументи в межах допустимого діапазону та дійсні вказані параметри.
- Ефективний. Ядра Micrium також включають в себе цінні статистичні дані про виконання, що робить інтерфейси вашої заявки видимими. Це

дає змогу виявити вузькі місця та оптимізувати використання енергії на початку свого циклу розробки.

Недоліки:

- Комерційна і немає безкоштовної версії.
- Важка у використанні.

ChibiOs / RT - операційна система реального часу. ChibiOs / RT призначена для вбудовування додатків, що працюють в реальному часі. Ця ОСРЧ відрізняється високою мобільністю, компактними розмірами і, головним чином має свою власну унікальну архітектуру, підходить для швидкого і ефективного перемикання контексту.

Функціонал застосування ChibiOs / RT:

- Ефективне і портативне ядро.
- Краща в класі перемикання контексту.
- Безліч підтримуваних архітектур і платформ.
- Статична архітектура.
- Динамічне розширення, динамічні об'єкти підтримуються за допомогою додаткового шару.
- Багатий набір примітивів для ОСРЧ: потоки, віртуальні таймери, семафори, м'ютекси, змінні умови / синхронізації, повідомлення , черги, прапори подій і поштовий ящик.
- Підтримка алгоритму успадкування для м'ютексів.
- HAL (Hardware Abstraction Layer - шар апаратних абстракцій), підтримка абстрактних платформ: GPIO, UART / USART, ADC, CAN, EXT, GPT, I2C, ICU, MAC, MMC, PWM, RTC, SDC, SPI, UART, USB, USB-CDC .
- Інструментарій для налагодження ОСРВ.

Ядро ChibiOs / RT було перенесено на безліч різних архітектур і для різних компіляторів, проте поточна стратегія полягає в підтримці меншого числа конкретних мікроконтролерів, і для поточного набору офіційно підтримуваних мікроконтролерів забезпечити повноцінний HAL. HAL

призначений для приховування відмінностей в апаратному забезпеченні від основної частини, алгоритму роботи програми, таким чином, щоб більша частина коду, що працює в режимі ядра, не потребувала в зміні при запуску ОСРЧ на різних платформах мікроконтролерів.

ChibiOS, як і будь-яка інша поважаюча себе RTOS має планувальник завдань з підтримкою витіснення і на даний момент має два варіанти функціонування:

- кооперативний розподіл часу між завданнями з одним пріоритетом
- Round-Robin планування завдань з однаковим пріоритетом із зазначенням кванта часу на завдання

У даний момент опції планувальника задаються глобально під час компіляції.

Для перемикання контексту, планувальником ядра використовується системний лічильник який також використовується для віртуальних таймерів.

Природно, будь-яке переривання може привести до перемикання контексту якщо в цьому є необхідність.

У ChibiOS / RT існують 3 класи логічних переривань:

- Регулярні переривання. Джерело переривань, що маскуються, переривання не можуть впливати на код ядра і таким чином з'являється можливість викликати API операційної системи зсередини обробників переривань. Обробники переривань, що належать до цього класу повинні бути описані за допомогою певних правил.
- Швидкі переривання. Джерело переривань, що маскуються з можливістю впливати на код ядра, таким чином мають менше прихованих станів роботи.
- Переривання, що не маскуються. Джерела переривань, що не маскуються контролюються операційною системою в повному обсязі і мають найменше число прихованих станів роботи.

Примітною особливістю ChibiOS можна назвати підтримку досить широкого спектра архітектур. Офіційно підтримувані архітектури: STM8, PPC , ARMv5-7 (LPC1x, LPC2x, STM32), MSP430, MegaAVR. Звичайно, цей список не може змагатися з FreeRTOS яка підтримує архітектуру більш ніж двадцяти двох процесорів і яку ми далі розглянемо.

QNX - операційна система реального часу, що призначена для роботи з вбудованими системами. Вважається однією з кращих реалізацій концепції мікроядерних операційних систем [9].

QNX це в першу чергу операційна система для побудови вбудованих систем. Але однією з основних особливостей QNX є модульність і, як наслідок, масштабованість [9]. Завдяки цьому QNX може застосовуватися і в мініатюрних контролерах, і в настільних комп'ютерах. А прозора мережа QNX дозволяє будувати продуктивні мережеві кластери. Оскільки QNX це мікроядерна операційна система, то основна ідея роботи її компонентів полягає в тому, що всі вони представлені як невеликі завдання, так звані сервіси. Тоді як в традиційних монолітних ядрах, ядро операційної системи представляє собою одну велику програму, що складається с великої кількості частин і кожна зі своїми особливостями. QNX дає можливість розробникам змінити функціональність виключивши непотрібне і при цьому не змінити ядро.

QNX складається з наступних двох основних компонентів:

- QNX Neutrino - включає в себе мікроядро (навіть набір мікроядер), драйвери, утиліти, графічне середовище і інше для підтримуваних апаратних платформ (x86, SH4, MIPS, PowerPC, ARM).
- QNX SDP (Software Development Platform) - включає в себе середовище розробки Momentics IDE на базі Eclipse, компілятори, заголовки та бібліотеки для розробки, а також QNX Neutrino [9].

QNX включає в себе наступні компоненти:

1. Адміністратор задач. В його функції входить запуск і закінчення завдань в системі та розподіл пам’яті.

2. Адміністратор периферійних пристроїв (Device Manager). Під його керівництвом знаходиться вся периферія, а саме : термінали, модеми, принтери, консоль і т.д. В його функції входить взаємодія з драйверами периферійних пристроїв. Також, адміністратор периферійних пристроїв несе відповідальність за допоміжні функції, наприклад стирання і відновлення рядків. Оскільки драйвери пристроїв для QNX є звичайним процесом то додавання нового драйвера ніяк не відбивається на функціонуванні системи.
3. Адміністратор файлової системи (Filesystem Manager). Його обов'язки полягають у підтримці файлової системи.
4. Мережевий Адміністратор (Network Manager). Відповідає за забезпечення комунікації в мережі. Його робота полягає у виконанні передачі повідомлень між процесами, що діють в різних вузлах мережі.

Головним недоліком QNX є те, що ця RTOS комерційна.

Одна з найпопулярніших ОСРЧ це FreeRTOS. FreeRTOS - багатозадачна операційна система реального часу (ОСРВ) для вбудованих систем [6].

Будь-яка , багатопотокова ОС, не буде вважатися повною, без відповідних засобів підтримки багатопотокового оточення. FreeRTOS володіє всім необхідним для цього, а саме:

1. Планувальник FreeRTOS підтримує три типи багатозадачності:
   - витісняючу;
   - кооперативну;
   - гібридну.

2. Розмір ядра FreeRTOS становить всього 4-9 кбайт, в залежності від типу платформи і налаштувань ядра.

3. FreeRTOS написана на мові С.

4. Підтримує завдання (tasks) і співпрограми (co-routines). Співпрограми спеціально створені для МК з малим об'ємом ОЗУ.

5. Багаті можливості трасування.

6. Можливість відстежувати факт переповнення стека.

7. Немає програмних обмежень на кількість одночасно виконуваних завдань.

8. Немає програмних обмежень на кількість пріоритетів завдань.

9. Немає обмежень у використанні пріоритетів: кільком завданням може бути призначений однаковий пріоритет.

10. Розвинені засоби синхронізації «задача - завдання »і« завдання - переривання »:

- черги;
- виконавчі семафори;
- рахункові семафори;
- рекурсивні семафори;
- м'ютекси.

11. М'ютекси з успадкуванням пріоритету.

12. Підтримка модуля захисту пам'яті (Memory protection unit, MPU) в процесорах Cortex-M3.

13. Поставляється з налагодженими прикладами проектів для кожного порту і для кожного середовища розробки.

14. FreeRTOS повністю безкоштовна.

До найголовніших переваг FreeRTOS можна віднести те, що вона безкоштовна, портована на велику кількість мікроконтроллерів, має широкий функціонал, велику кількість бібліотек і гарну документацію тому це чудове рішення для реалізації в будь-кому проекті. Для свого проекту я обрала саме цю операційну систему .

3.ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПІДПРОГРАМ

Після того як було поставлено задачу та розглянуто існуючі рішення можна приступити до етапу розробки , який включає побудову архітектури програми відповідно до поставлених вимог. Спочатку доцільно розглянути архітектуру програмного забезпечення.

3.1 Аналіз вимог до функціональності програми

Перед початком розробки було встановлено наступні вимоги до програмного забезпечення:

1. Передача і прийом даних по лінії живлення.
2. Обробка всіх помилок зв’язку.
3. Швидкодія виконання передачі і прийому даних.
4. Портування програмного забезпечення на різні мікроконтролери.

3.2 Цілісність даних при передачі інформації.

При передачі інформації між пристроями виникають часові затримки, що впливають на швидкість обміну та реакції пристроїв. Затримки виникають через наступні фактори:

- Довгі лінії
- Вплив завад
- Відбиття сигналу (неузгоджене навантаження лінії)

Індуктивність та ємність лінії впливає на цілісність сигналу що може призводити до збоїв або некоректної передачі даних. Для зменшення впливу довжини лінії на кінцях дротів має бути встановлене узгоджувальне навантаження.

Лінія що має велику індуктивність випромінює більше імпульсних завад під час перехідних процесів. Для зменшення завад модулюючий сигнал повинен мати якомога менше складових гармонік, тому при передачі даних по лінії живлення застосовуються модулюючі сигналі близькі до синусоїди.

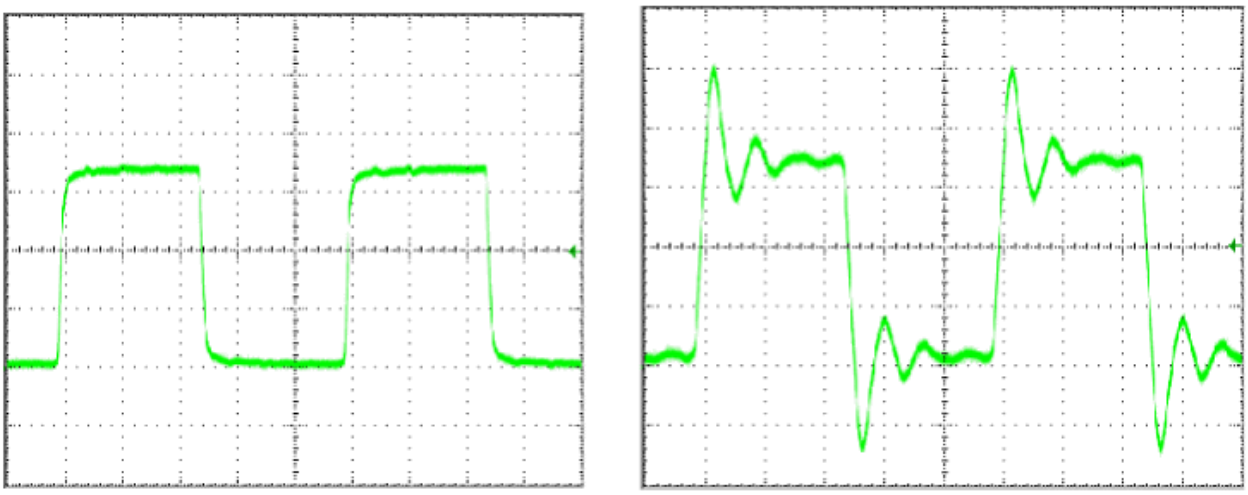


Рисунок 3.1- а) Форма сигналу з узгодженням навантажень, б) Форма сигналу на неузгодженій лінії

Цілісність передачі інформації можна забезпечити спеціалізованим протоколом зв’язку з використанням цифрової модуляції та вирахуванням контрольної суми.

Контрольна сума являє собою деяке значення яке передавач підраховує і додає його до повідомлення. Це значення побудоване за певним алгоритмом на основі кодованого повідомлення. Перевірочна інформація додається до основного повідомлення в кінець після корисних даних.

Приймач використовує ту ж функцію для підрахунку контрольної суми, і порівнює отримане значення з записаним у кінці повідомлення. При передачі повідомлень по реальних каналах зв’язку, можливо внесення спотворень в ці повідомлення, і тому необхідно передбачити методи, за допомогою яких приймач зможе визначити, чи були помилки при передачі.

Використання алгоритмів підрахунку контрольної дає змогу побачити різницю між вихідними даними і даними на прийомнику. При виявленні помилок, приймач може запросити повторну передачу повідомлення.

На даний момент існує безліч алгоритмів обчислення контрольної суми: додавання байтів, CRC,MD5, SHA і інше. CRC традиційно використовується в провідних і безпровідних протоколах передачі даних (Ethernet. Bluetooth,

ZigBee, CAN, Fibre Chanel) для контролю цілісності фрагментів чи кадрів даних.

```
unsigned char Crc8(unsigned char *pcBlock, unsigned int len)
{
    unsigned char crc = 0xFF;
    unsigned int i;
    while (len--)
    {
        crc ^= *pcBlock++;
        for (i = 0; i < 8; i++)
            crc = crc & 0x80 ? (crc << 1) ^ 0x31 : crc << 1;
    }
    return crc;
}
```

Рисунок 3.2- Приклад обчислення контрольної суми за алгоритмом CRC-8

3.3 Затримки при передачі інформації.

Додаткові затримки виникають саме в кільцевій топології зв'язку. Ці затримки пов'язані зі швидкістю обробки даних та відсутністю правильного ретранслювання.

У кільцевому інтерфейсі усі зв'язки одиночні та не мають розгалужень. При передачі інформації до кінцевого отримувача, інформацію спочатку отримає посередній пристрій та направить далі по кільцю до отримувача.

На рисунку 3.3 показано діаграму передачі даних по кільцю з демонстрацією затримок. П1 ініціює передачу, П2 та П3 транслюють дані, П4 отримує данні.

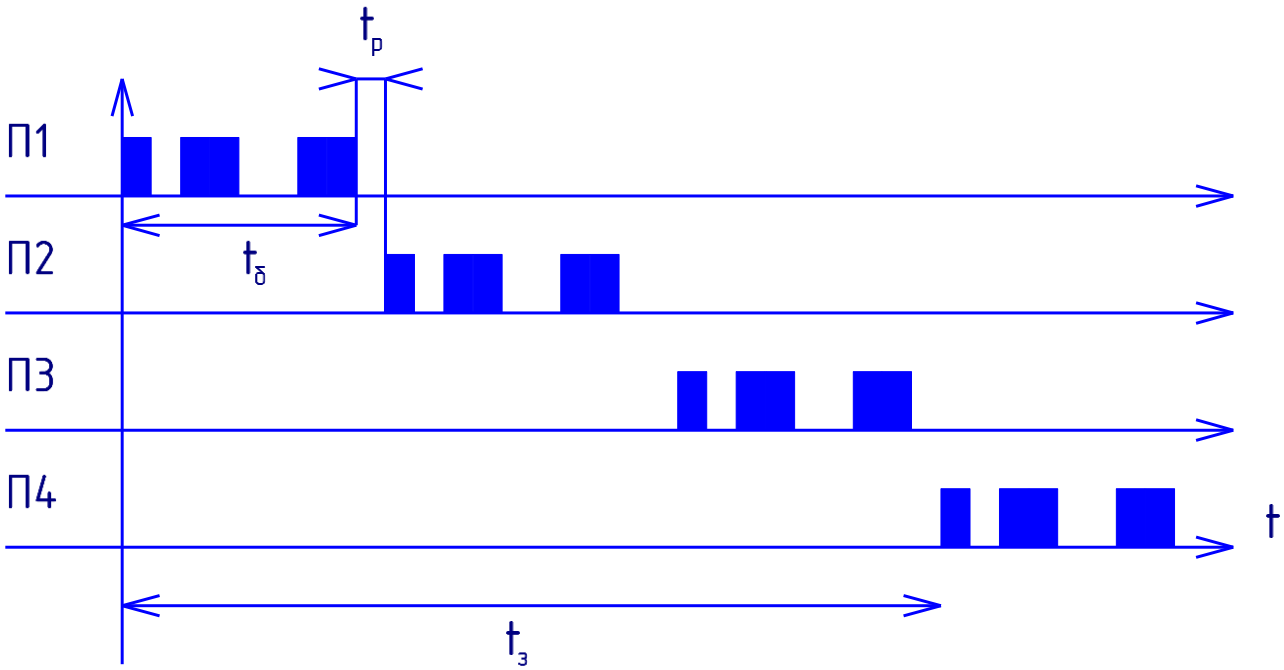


Рисунок 3.3 - Затримки при звичайній ретрансляції

Де T_p – затримка реакції

$T_б$ – період передачі байту

$T_з$ – повна затримка сигналу

З даної діаграми видно, що основну затримку створює ретранслювання даних по байтам або по пакетах. Зі збільшенням пристроїв повна затримка сигналу буде тільки збільшуватись $T_з = (T_p + T_б) * N$, де N – кількість проміжних пристроїв . Це створює дуже багато проблем коли необхідна швидка реакція системи.

Запропонований мною метод ретрансляції зменує затримки обміну даними за допомогою побітної ретрансляції. Метод полягає у тому, що дані ретранслюються одразу з мінімальною затримкою яка залежить від швидкості виконавчого пристрою. В такому випадку кільцевий інтерфейс конфігурується так, щоб утворити «шину» для даних, при цьому зберігаючи усі функційні можливості «кільцевого» включення. На рисунку показано діаграму передачі даних з побітною ретрансляцією

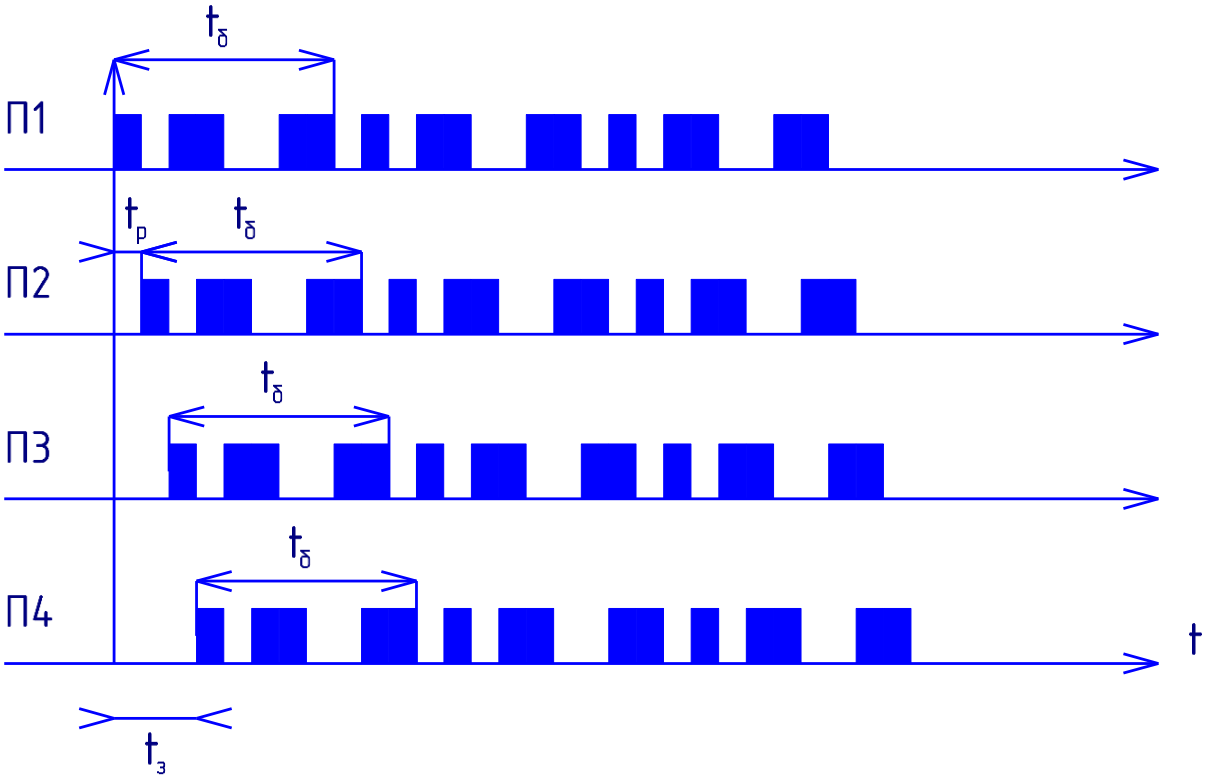


Рисунок 3.4 - Затримки при побітній ретрансляції

З цього випливає що час повної затримки сигналу залежить тільки від швидкості виконавчого пристрою, $T_з = T_p * N$.

3.4 Забезпечення надійності передачі інформації та огляд протоколу обміну між керуючими пристроями.

В якості програмного протоколу обміну даними буде використовуватись модифікований протокол MODBUS. Це комунікаційний протокол заснований на технології головний-підлеглий. Набув широкого використання в промисловості де потрібна організація зв’язку між електронними пристроями. MODBUS використовують для передачі інформації через послідовні лінії зв'язку RS-485, RS-422, RS-232, а також мережі TCP/IP.

Контролери на шині Modbus взаємодіють, використовуючи модель головний-підлеглий, засновану на транзакціях, що складаються із запиту і відповіді.

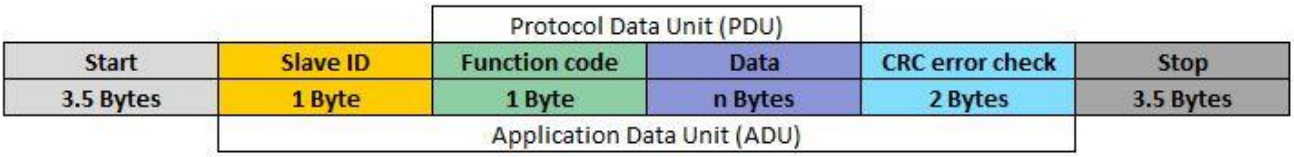


Рисунок 3.5 – Блок даних протоколу Modbus

Зазвичай в мережі є тільки один клієнт, так званий «головний» (англ. master) пристрій, і кілька серверів - «підлеглих» (англ. slaves) пристроїв. Головний пристрій є ініціатором транзакцій (передає запити). Підлеглі пристрої передають запитувані головним пристроєм дані, або виконують запитувані дії.

В мережах MODBUS може бути використаний один з двох способів передачі: ASCII або RTU. Користувач обирає необхідний режим разом з іншими параметрами (швидкість передачі, режим паритету і т. Д.) Під час конфігурації кожного контролера.

Режим ASCII:

- Формат бітів: 1 старт біт;
- 7 біт даних, молодшим бітом вперед;
- 1 біт паритету або без паритету;
- 1 стоп біт якщо є паритет; 2 біта якщо нема паритету;
- Контрольна сума: Longitudinal Redundancy Chek (LRC).

При використанні ASCII режиму кожен байт повідомлення містить два 4-х бітних шістнадцяткових числа.

Кожне повідомлення передається безперервним потоком. Формат кожного байта в ASCII режимі:

Start	Address	Function	Data	LRC	End
1 char :	2 chars	2 chars	0 up to 2x252 char(s)	2 chars	2 chars CR,LF

Рисунок 3.6 - Блок даних протоколу Modbus в режимі ASCII

Система кодування: Шістнадцятирічна, ASCII-символи 0-9, A-F.

Режим RTU**:** система кодування: 8 бітова двійкова, шістнадцяткова 0-9, A-F. Призначення бітів:

- 1 старт біт;
- 8 біт даних, молодшим значущим розрядом вперед;
- 1 біт паритету або без паритету ;
- 1 стоп біт якщо є паритет; 2 біта якщо немає паритету;
- Контрольна сума: Cyclical Redundancy Check (CRC).

3.5 Аналіз архітектури прогамного забезпечення мікроконтролера

Як вже було акцентовано раніше, налаштовування мікроконтролера STM32F0 буде відбуватися за допомогою STM32CubeMX.

В результаті роботи з STM32CubeMX ми отримуємо проект з прописаними налаштуваннями, підключеними бібліотеками та з кодом ініціалізації периферії мікроконтролера. Структура файлів проекту виглядає так:

- Drivers/STM32F0xx_HAL_Driver- функції HAL рівня;
- Drivers/CMSIS - драйвери низького рівня;
- Application / USER - прикладні функції (main.c, _it.c - опис оброблювачів переривання і ін.);
- Middlewares/FreeRTOS – файли операційної системи;
- Application / MDK-ARM - тут стартап асемблерний код (файл startup_stm32f030x6.s), в якому вказано опис функцій обробників переривання і вказівка на початкову вхідну функцію, тобто функцію main, з якої буде відбуватися завантаження проекту.

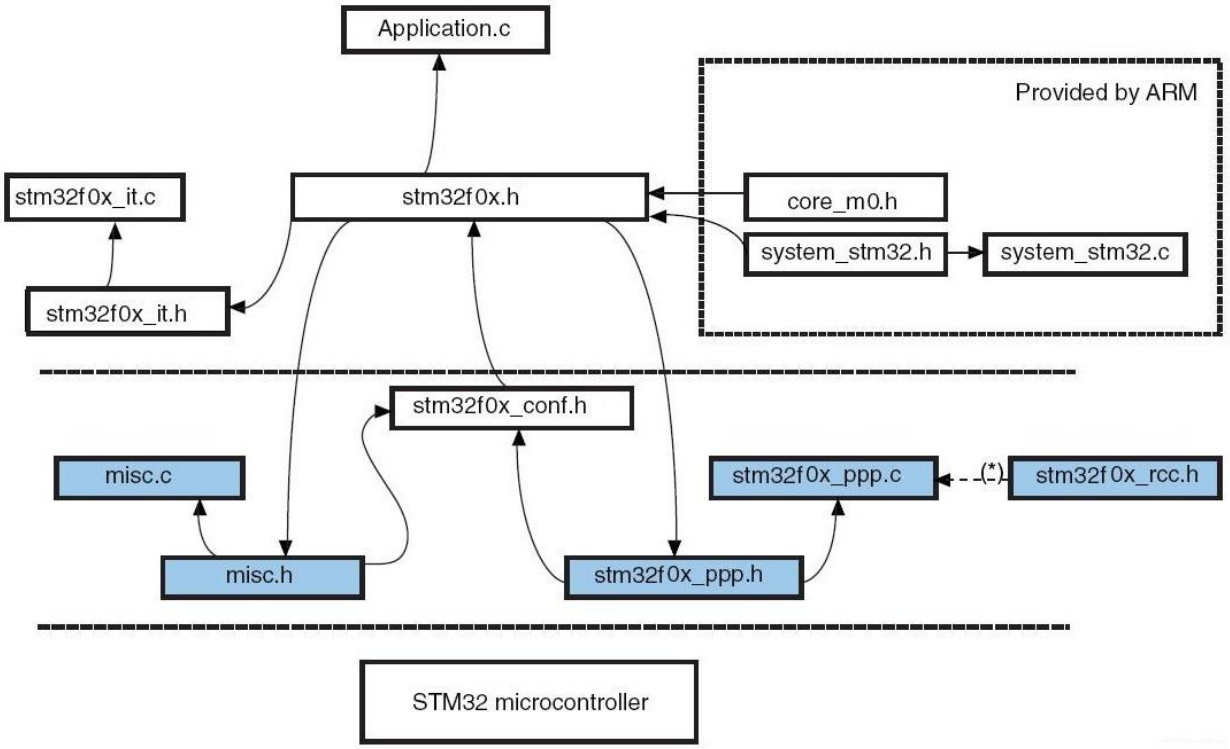


Рисунок 3.7 - Архітектура стандартних бібліотек периферії для STM32

Структура бібліотеки складається з двох взаємодоповнюючих складових. Перша складова – заголовочні файли та файли реалізації всієї периферії мікроконтролерів STM32 - STM32F0xx_StdPeriph_Driver. Вся функціональність периферійних модулів описана в заголовних файлах і файлах реалізації [10]. Наприклад, для портів введення-виведення це два файли - stm32f0xx_gpio.h і stm32f0xx_gpio.c.

Друга складова – заголовочні файли та файли реалізації самого ядра ARM Cortex-M0 від компанії ARM - CMSIS (ARM CortexTM Microcontroller Software Interface Standard) [10]. Ядро ARM Cortex-M0 виходить за рамки звичайного поняття ядра мікроконтролера і являє собою міні-мікроконтролер з периферією - вбудований системний таймер, контролер переривань і т.д. CMSIS представляє собою константи і визначення, функції доступу до регістрів і периферійних модулів ядра, незалежний інтерфейс для операційних систем реального часу (RTOS). CMSIS складається з трьох файлів:

- core_m0.h - допоміжні функції доступу до регістрів ядра;

- startup_stm32f0x_xx.s- набір файлів для кожної лінійки сімейства STM32, що забезпечують ініціалізацію стека і таблицю векторів переривань;
- system_stm32f0xx.h- файл початкової ініціалізації тактової частоти мікроконтролера.

Також надаються три файли доступні для модифікації користувачем:

- stm32f10x_conf.h - файл конфігурації бібліотеки. Користувач може підключити або відключити модулі.
- stm32f10x_it.h – заголовний файл, що включає в себе всі можливі обробники переривань (їх прототипи).
- stm32f10x_it.c - шаблонний файл вихідного коду, що містить сервісні рутинні переривання (interrupt service routine, ISR) для виняткових ситуацій в CortexM0. Користувач може додати свої ISR для використовуваної периферії.

Для програмування мікроконтролера ми будемо використовувати базову бібліотеку STM32 HAL Driver. STM32Cube HAL - вбудоване програмне забезпечення рівня абстракції HAL, для STM32, що забезпечує максимальну переносимість коду всередині сімейства STM32 [7]. Драйвери рівня HAL являють собою комплект універсальних, багатофункціональних, і одночасно простих інтерфейсів API, призначених для взаємодії МК з верхнім шаром ПЗ (основною програмою, бібліотеками і стеками). Драйвери можуть мати як загальний , так і розширений API.

HAL розроблений для застосування такої архітектури програмування, коли необхідні функції виконуються верхнім шаром додатку, за рахунок застосування проміжного рівня CMSIS. При такій архітектурі програмування верхній рівень програми не «прив'язаний» до мікроконтролера. Така структура користувацького додатку покращує повторне використання коду, і гарантує його легку переносимість на інші пристрої STM32.

API-інтерфейси драйверів HAL, діляться на дві категорії:

1) Загальні (generic) API, які забезпечують загальні, для всіх серій STM32, функції.

2) Розширені (extension) API, які містять специфічні або індивідуальні функції для даного сімейства або його частини.

Драйвери HAL реалізують виявлення помилок під час виконання (run-time failure detection). Така динамічна перевірка сприяє підвищенню надійності вбудованого ПО. Виявлення помилок під час виконання програми, також, сприяє прискоренню розробки призначених для користувача додатків і процесу налагодження.

3.6 Методи поліпшення безпеки і надійності програми

Для поліпшення безпеки, переносимості і надійність програм для вбудованих систем використовується стандарт розробки програмного забезпечення Misra C. Мова С є найбільш популярною мовою програмування високого рівня для вбудованих систем. Однак при розробці коду для систем, вимогливих до безпеки мова С виявляє багато недоліків. Існує кілька невизначених або реалізаційно-залежних аспектів мови С, що роблять його невідповідним для розробки систем, вимогливих до безпеки.

Асоціація надійності програмного забезпечення для автомобілебудування в Великобританії (MISRA) зрозуміла, що в багатьох областях автомобільного дизайну безпека має першорядне значення. Вони також визнали, що С є де-факто мовою для кодування таких систем, і тому ці системи є вразливими до обмежень С. Замість того, щоб передбачати використання безпечної мови, такої як Ada, організація переглянула способи зробити С безпечнішим. Тому у 1998 році Асоціація надійності програмного забезпечення для автомобілебудування у Великобританії встановила 127 директив про використання С у критично важливих системах. Всього MISRA C має 127 правил. З них 93 необхідні, а решта 34 - консультативні. Різниця між цими двома типами правил є важливою.

У 2004 році було зроблено друге видання "Керівні принципи використання мови С у критичних системах" або MISRA-C: 2004 з багатьма істотними змінами до керівних принципів, включаючи повну перенумерацію правил. MISRA-C: 2004 містить 142 правила, з яких 122 є "обов'язковими", а 20 - "консультативні"; вони поділяються на 21 актуальну категорію, від "Середовища" до "Помилки виконання часу". У 2012 році було зроблено видання MISRA-C: 2012 яке містило 143 правила та 16 "директив" (тобто правила, відповідність яких є більш відкритою для інтерпретації або вони стосуються процесуальних чи процедурних питань); Правила діляться на обов'язкові, необхідні і рекомендаційні; можуть поширюватися на окремі одиниці трансляції або на всю систему. Також правила розділені на Decidable і Undecidable.

MISRA C нічого не робить для перевірки правильності алгоритму, не забезпечує виконання певного стилю програмування, і не зупинить від написання непрацюючого коду. Але вона може захистити програміста від більшості темних кутів мови С.

3.7 Огляд цифрової частини інтерфейсу зв’язку

Універсальний асинхронний приймач-передавач (UART) приймає байти даних і передає окремі біти послідовним чином. UART приймача повторно збирає біти в повні байти. Послідовна передача цифрових даних (біт) за допомогою одного кабелю або іншого носія менш затратна, ніж паралельна передача через кілька проводів.

UART зазвичай не генерує та не отримує зовнішні сигнали, що використовуються між різними пристроями. Окремі інтерфейсні пристрої використовуються для перетворення сигналів логічного рівня UART на рівні зовнішнього сигналу, що може бути стандартизованими рівнями напруги, рівнями струму або іншими сигналами.

Основні робочі лінії у нас - RXD і TXD, або просто RX і TX. Лінія передатчик - TXD (Transmitted Data), а порт RXD (Received Data) - приймаюча. Вихід передавача TX з'єднаний з входом приймача RX і навпаки [10].

Передача даних в UART здійснюється по одному біту в рівні проміжки часу. Цей часовий проміжок визначається заданою швидкістю UART і для конкретного з'єднання вказується в бодах (що в даному випадку відповідає бітам в секунду). Існує загальноприйнятий ряд стандартних швидкостей: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бод [10]. Швидкість (S, бод) і тривалість біта (T, секунд) пов'язані співвідношенням T = 1 / S. Швидкість в бодах іноді називають сленговим словом бодрейт або бітрейт.

Крім інформаційних бітів, UART автоматично вставляє в потік мітки синхронізації, так звані стартовий і стоповий біти. При прийомі ці зайві біти видаляються з потоку. Зазвичай стартовий і стоповий біти обрамляють один байт інформації (8 біт), при цьому молодший інформаційний біт передається першим, відразу після стартового. Зустрічаються реалізації UART, що передають по 5, 6, 7, або 9 інформаційних біт. Обрамлені стартом і стопом біти є мінімальною посилкою. Деякі реалізації UART використовують два стопових біти при передачі для зменшення ймовірності розсинхронізації приймача і передавача при щільному трафіку. Приймач ігнорує другий стоповий біт, сприймаючи його як коротку паузу на лінії. На рисунку 3.8 можна побачити схему представлення передачі байду даних.

Рисунок 3.8- Представлення передачі байту даних

Зв'язок може бути симплекс (тільки в одному напрямку), повний дуплекс (обидва пристрої одночасно надсилають та приймають) або напівдуплекс (пристрої по черзі передають та отримують)

Для реалізації напівдуплексного зв’язку у кільцевому інтерфейсі необхідно мати 2 таких UART у кожному пристрої. Можна стикнутись з проблемою наявності такої кількості апаратних інтерфейсів UART. Зазвичай у простих та дешевих контролерах такий інтерфейс буває тільки один. В такому випадку вирішити проблему можна двома шляхами:

- Використовувати мультиплексор
- Реалізувати програмний UART

Перший варіант є простим, але створює ряд незручностей у керуванні та плануванні зв’язку. Натомість другий варіант не обмежує зв'язок, але потребує більше ресурсів процесора.

3.8 Розробка концепції рішення поставлених вимог

Для реалізації програмного інтерфейсу UART достатньо реалізувати прості функції обміну.

Дані про UART будуть зберігатися в полях структури SoftwareUartInterface. Детально розглянемо поля цієї структури :

- void (* **const** TxWritePin)(uint8_t state) - вказівник на функцію запису в пін значення (HAL);
- uint8_t (* **const** ReadRxPin)(void) - вказівник на функцію зчитування з піну значення (HAL);
- void (* **const** Delay)(MCU_TIM_TYPE delay) - вказівник на функцію затримок (API);
- void (* **const** AppendByte)(uint8_t byte) - вказівник на функцію зберігання отриманого біту (API);
- void (* **const** SetRxExtiState)(uint8_t state) - вказівник на функцію дозволу переривання по піну RX (API);

- **volatile** MCU_POINTER_TYPE * **const** TimerCounterPtr - вказівник на лічильник таймера(API).

```
struct SoftwareUartInterface {
    void (* const TxWritePin)(uint8_t state);
    uint8_t (* const ReadRxPin )(void);
    void (* const Delay)(MCU_TIM_TYPE delay);
    void (* const AppendByte)(uint8_t byte);
    void (* const SetRxExtiState)(uint8_t state);
    volatile MCU_POINTER_TYPE * const TimerCounterPtr;
};
```

Всі дані про UART містяться в структурі SoftwareUart_t. Розглянемо детально поля структури :

- Interface – містить API та HAL функції для роботи з мікроконтролером. (поле користувача).
- Scheduler - планувальник задач на перериваннях. (поле користувача). TxBitsCounter - лічильник бітів передавача (приватне поле).
- TxBits – байт даних передавача (приватне поле).
- RxBitsCounter – лічильник бітів приймача (приватне поле).
- RxBits – байт даних приймача (приватне поле).
- BaudRate – частота передачі даних (поле користувача).
- TxDataBuffer – буфер передачі даних (приватне поле).
- TxDataCounter – лічильник передачі байтів (приватне поле).
- RxErrorCode – змінна що зберігає інформацію про помилки приймача (поле користувача).
- TxErrorCode – змінна що зберігає інформацію про помилки передавача (поле користувача).
- DataSize – формат передачі даних, кількість біт (поле користувача).
- StopBits – формат передачі даних, кількість стоп біт (поле користувача).
- TransmitterState – кінцевий автомат станів передавача (приватне поле).
- ReceiverState – кінцевий автомат станів приймача (приватне поле).

```
typedef struct SoftwareUart_t {
```

```
    struct SoftwareUartInterface*   Interface;
    ItScheduler*                Scheduler;
    uint16_t                        ProcessTick;
    uint16_t                        TxBitsCounter;
    uint16_t                        TxBits;
    uint16_t                        RxBitsCounter;
    uint16_t                        RxBits;
    uint16_t                        BaudRate;
    uint8_t*                        TxDataBuffer;
    uint8_t                         TxDataCounter;
    uint8_t                         RxErrorCode;
    uint8_t                         TxErrorCode;
    SoftwareUartDataSize            DataSize;
    SoftwareUartStopBits            StopBits;
    SoftwareUartTransmitterState    TransmitterState;
    SoftwareUartReceiverState ReceiverState;
} SoftwareUart;
```

Функція SoftwareUartTransmit(SoftwareUart* suart, uint8_t* data, uint16_t size) виконує відправку пакету даних data з заданим розміром size через інтерфейс suart з використанням затримок. На рисунку 3.9 зображено алгоритми роботи функцій передачі і прийому даних без використання переривань і планувальника.

Функція SoftwareUartTransmitByte(SoftwareUart* suart, uint8_t byte) виконує передачу байту через інтерфейс suart. Відправка відбувається з використанням затримок.

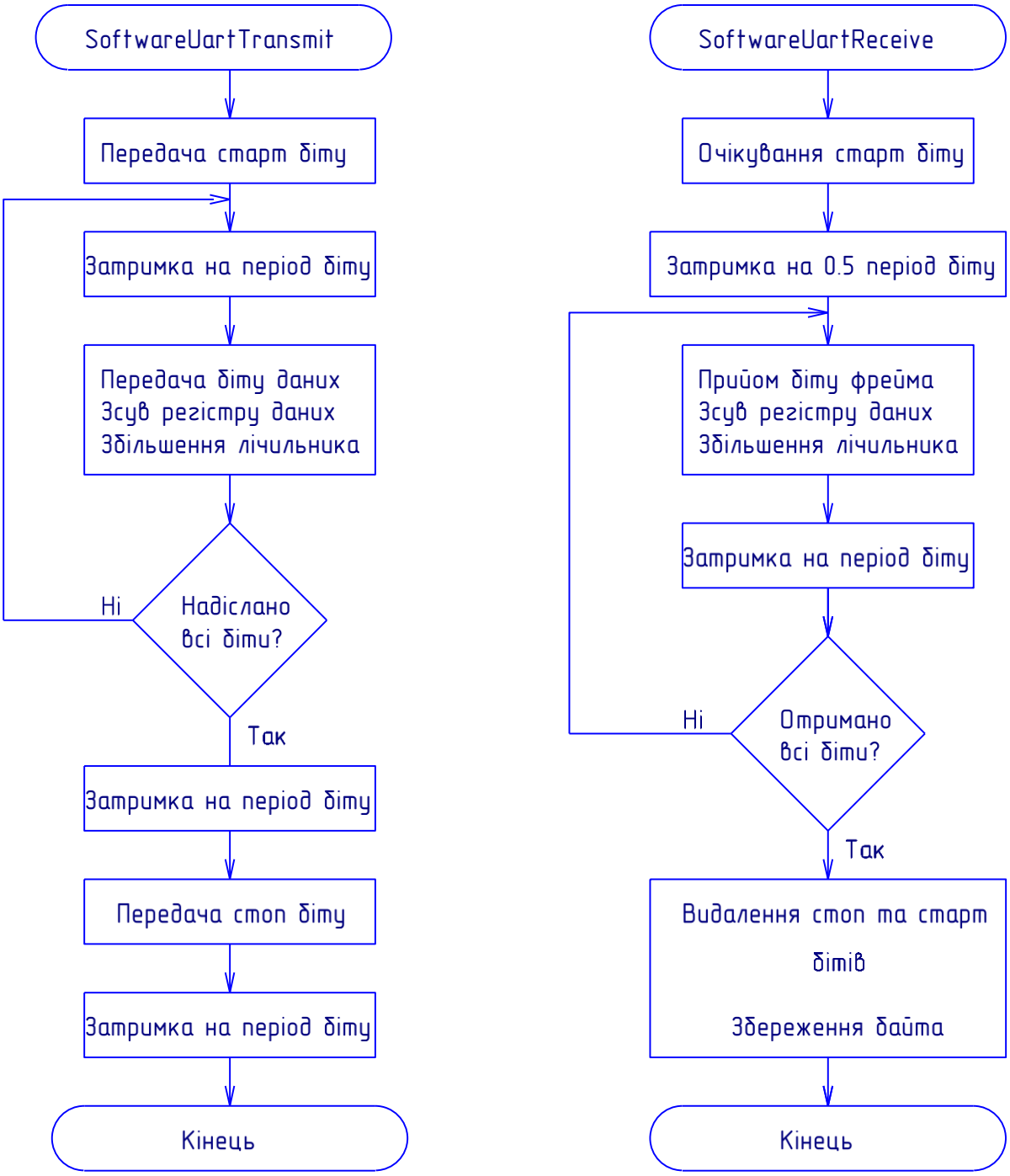


Рисунок 3.9 – Алгоритми роботи функцій SoftwareUartTransmit і SoftwareUartReceive

Ці функції викликаються при передачі та прийомі інформації. Але в них є досить важливий недолік – функції блокують роботу основної частини програми в процесі очікування затримок. При цьому функція більшу частину часу просто чекає моменту передачі чи прийому наступного біта. Через це можливість одночасно приймати і ретранслювати дані зникає.

Для розв’язання цієї проблеми необхідно забезпечити роботу функцій прийому та передачі без затримок та блокування. Рішення – реалізація програмного інтерфейсу UART на перериваннях. Для цього необхідно створити планувальник переривань з чергами та пріоритетами.

Планувальник програмних переривань слугує для запуску деяких підпрограм (функцій) в певні , зазначені проміжки часу. Планувальник використовує апаратний або програмний таймер та вектор переривань по досягненню певного значення (Capture Compare Interrupt) та переривання по переповненню (Update Interrupt).

Таймери - це периферія контролера STM32, що дозволяє точно відраховувати інтервали часу. STM32F0 містить 11 різноманітних таймерів розрядністю 16 або 32 біта: таймер з розширеними функціями (управління двигунами), таймери загального призначення, два сторожових таймера (незалежний і віконного типу) і 24-розрядний системний таймер.

Таймери поділяються на три групи: базові (basic timers), загального призначення (general-purpose timers) і просунуті (advanced-control timers).

Таблиця 3.1. Порівняння характеристик таймерів в STM32F030

Тип таймеру	Таймер	Розширення лічильника	Напрям рахунку	Значення передпільника	Канали захоплення/порівняння
Advanced control	TIM1	16-bit	Вверх, вниз, вверх/вниз	Будь-яке число від 1 до 65536	4
General purpose	TIM3		Вверх, вниз, вверх/вниз		4
	TIM14		Вверх		1
	TIM16, TIM17		Вверх		1

Таймери загального призначення дозволяють генерувати переривання при виникненні наступних подій:

- переповнення / обнулення або ініціалізація лічильника;
- при виникненні такої події як пуск, зупинка, ініціалізація або рахунок лічильника від зовнішнього / внутрішнього сигналу;
- при захопленні по входу;
- при порівнянні. Також є можливість підключити інкрементний енкодер і датчик Холла.

За роботу таймерів відповідають такі регістри:

Базові регістри лічильника (x - 2 або 3, в залежності від таймера):

- Лічильний регістр (TIMx_CNT) TIM2 and TIM3 counter
- Регістр переддільника (TIMx_PSC)
- Регістр автоперезапуску (TIMx_ARR) TIM2 and TIM3 auto-reload register

Таймер відраховує певні проміжки часу та при досягненні потрібного моменту часу відбувається переривання програми і викликається обробник переривання. Обробник виконує певну підзадачу та передає управління основній програмі. Для досягнення максимальної ефективності виконання підзадач не повинно займати великі проміжки часу. Бібліотека передачі даних по UART написана так, що її можливо використовувати з будь-яким таймером з периферії STM32.

Вся інформація про планувальник буде знаходитися в структурі ItScheduler. Ця структура міститиме поля :

- ItSchedulerInterface* Interface де тип даних користувача ItSchedulerInterface це структура, що містить поля :
    1. TimerCounterPtr вказівник на регістр лічильника таймера.
    2. TimerCCRPtr вказівник на регістр захоплення/порівняння.
    3. void (*const ClearItFlag)(void) вказівник на функцію, що очищує флаги переривань.
    4. void (*const SetCounterState)(uint8_t state) вказівник на функцію, що вмикає/вимикає лічильник таймера.

5. void (*const SetItState)(uint8_t state) вказівник на функцію, що дає/не дає дозвіл на виклик переривання.

- ItTask Tasks[SCHEDULER_QUEUE_SIZE] – масив з задачами планувальника.
- TaskCount – лічильник задач планувальника.

```
typedef struct ItSchedulerInterface_t {
volatile MCU_POINTER_TYPE * const   TimerCounterPtr;
volatile MCU_POINTER_TYPE * const   TimerCCRPtr;
void    (*const ClearItFlag)(void);
void    (*const SetCounterState)(uint8_t state);
void    (*const SetItState)(uint8_t state);
} ItSchedulerInterface;

typedef struct ItScheduler_t {
         ItSchedulerInterface* Interface;
         ItTask  Tasks[SCHEDULER_QUEUE_SIZE];
         uint8_t TaskCount;
     } ItScheduler;
```

Рисунок 3.10 – Структури, що зберігають дані про планувальник

Функція SchedulerAddTask(ItScheduler* scheduler, ItTask task) використовується для додавання нових задач в планувальник. Для додавання нової задачі в масив задач планувальника потрібно знайти їй місце беручи до уваги її пріоритет. Як тільки знайдено місце для нової задачі відбувається зсув всіх інших задач для того, щоб звільнити місце для нової. Слід зазначити, що під час виконання цих дій відповідальний таймер призупиняється і поновлює свою роботу тільки коли всі дії по додаванню нової задачі виконані. Алгоритми роботи цієї функціїї зображено на рисунку 3.10.

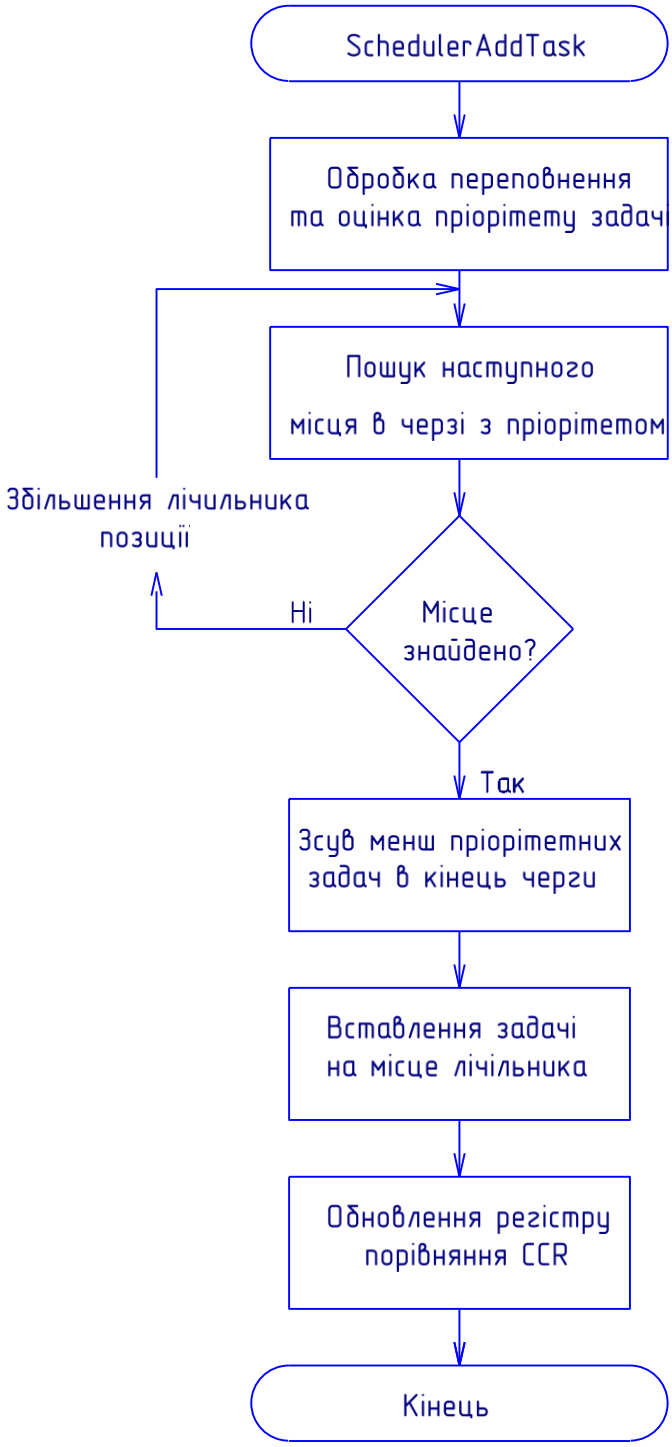


Рисунок 3.11 – Алгоритми роботи функцій ScedulerAddTask

Функція SchedulerCCISR(ItScheduler* scheduler) – оброблює переривання по порівнянню. Кожний раз коли відбувається переривання по порівнянню планувальник віддає на виконання задачу з найвищим пріоритетом. Всі інші задачі в масиві зсуваються на одну комірку вліво змінюючи тим самим свій пріоритет. Далі відбувається виклик задачі для відпрацювання. Під час виконання цих дій свою роботу призупиняє основний

таймер, щоб уникнути накладання викликів задач. Алгоритми роботи цієї функціїї зображено на рис. 3.11.

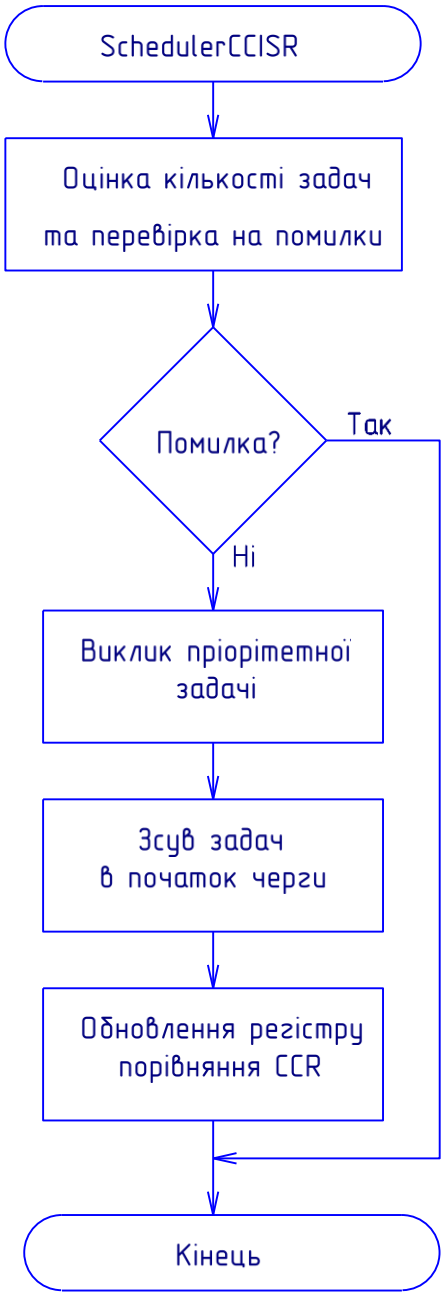


Рисунок 3.12 – Алгоритми роботи функцій ScedulerCCISR

Функція SchedulerUpdateISR(ItScheduler* scheduler) - обробник переривань по переповненню. Виконує обробку пріоритетів задач для нового циклу таймера. Алгоритми роботи цієї функціїї зображено на рис. 3.12.

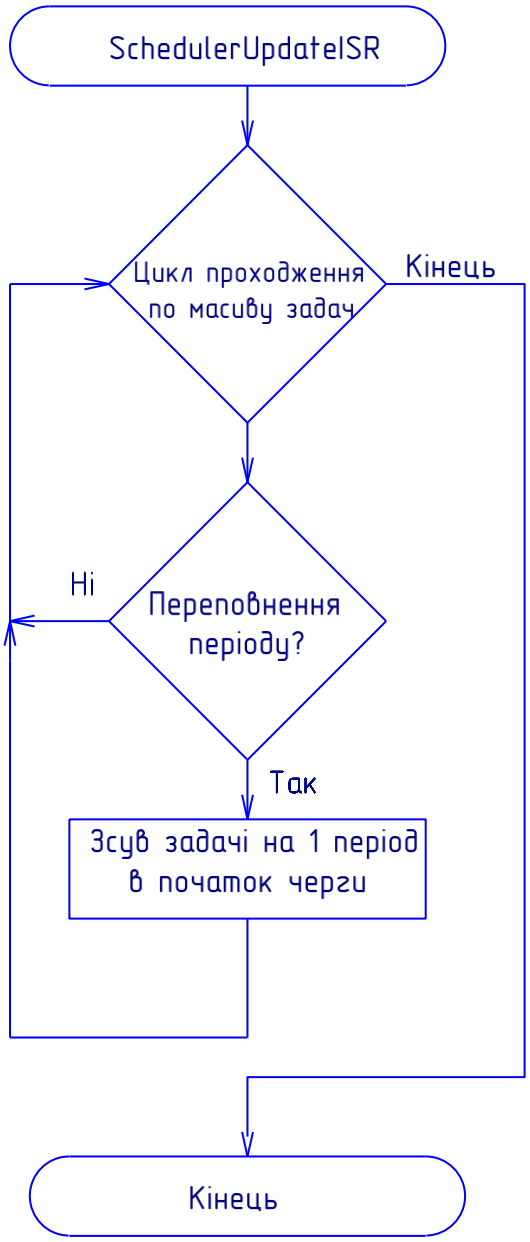


Рисунок 3.13 – Алгоритми роботи функцій ScedulerUpdateISR

Функція xAddItTaskInTick(ItScheduler* scheduler, void (*func)(), void* param, uint16_t tick) – додає в чергу задачу, яка має виконуватися коли лічильник таймера буде рівний параметру tick.

Функція xAddItTaskInDelay(ItScheduler* scheduler, void (*func)(), void* param, uint16_t tick) – додає в чергу задачу , яка виконається через певний проміжок часу.

Функція CopyTask(ItTask* target, ItTask* source) – копіює задачу source в target.

Функція ClearTask(ItTask* target) – видаляє задачу з масива планувальника.

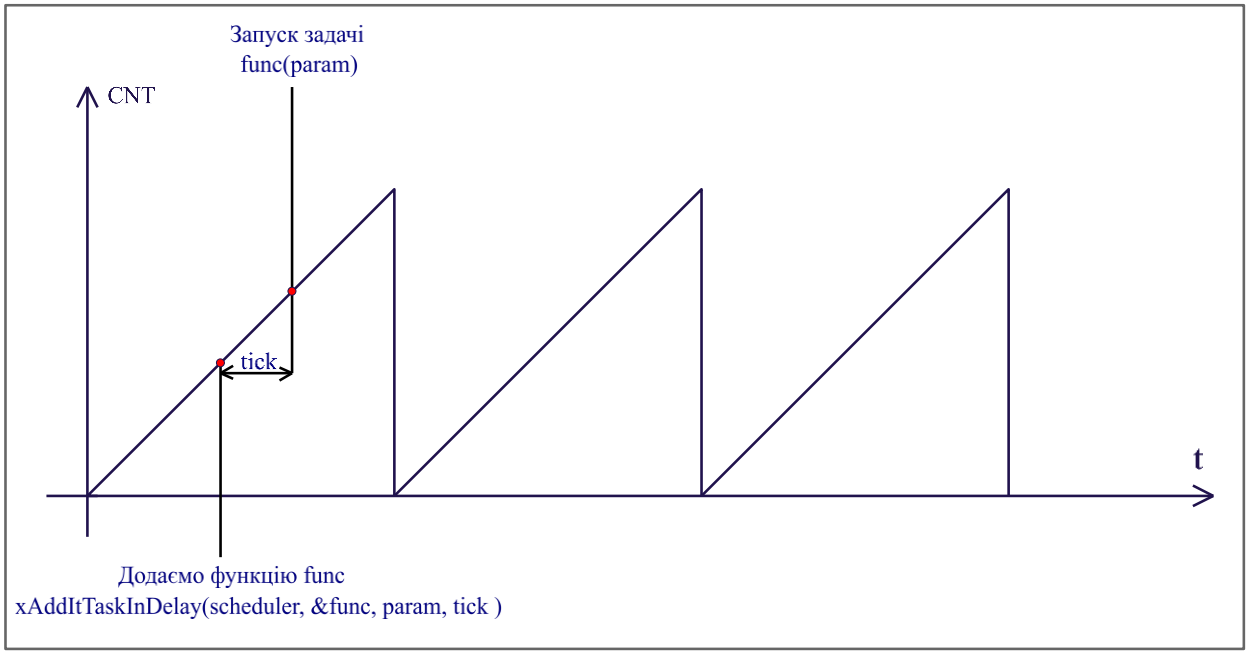


Рисунок 3.14 – Графічне відображення принципу роботи функції AddItTaskInDelay().

Функції для роботи з UART побудовані на перериваннях і з використанням планувальника.

```
SoftwareUartStatus SoftwareUartTransmitIt(SoftwareUart* suart, uint8_t* data, uint16_t size);
```

Функція виконує відправку пакету даних data з заданим розміром size через інтерфейс suart по перериваннях. Використовує інтерфейси планувальника та апаратного таймера. Передача даних блокує інші переривання в момент виконання (критична секція). Повертає результат статусу виконання функції.

```
SoftwareUartStatus SoftwareUartRxSetCmd(SoftwareUart* suart, uint8_t cmd);
```

Функціє дозволяє, або забороняє прийом даних інтерфейсу suart по команді cmd . Вмикає\вимикає переривання по прийому та забороняє виконання приватних функцій прийому. Прийом даних блокує інші переривання в момент виконання (критична секція). Повертає результат статусу виконання функції.

```
   SoftwareUartStatus      SoftwareUartRxIRQHandler(SoftwareUart*
suart);
```

Функція повинна викликатись у векторі переривання. Оброблює переривання по зміні стану піна RX. Слугує для прийому даних. Повертає результат статусу виконання функції.

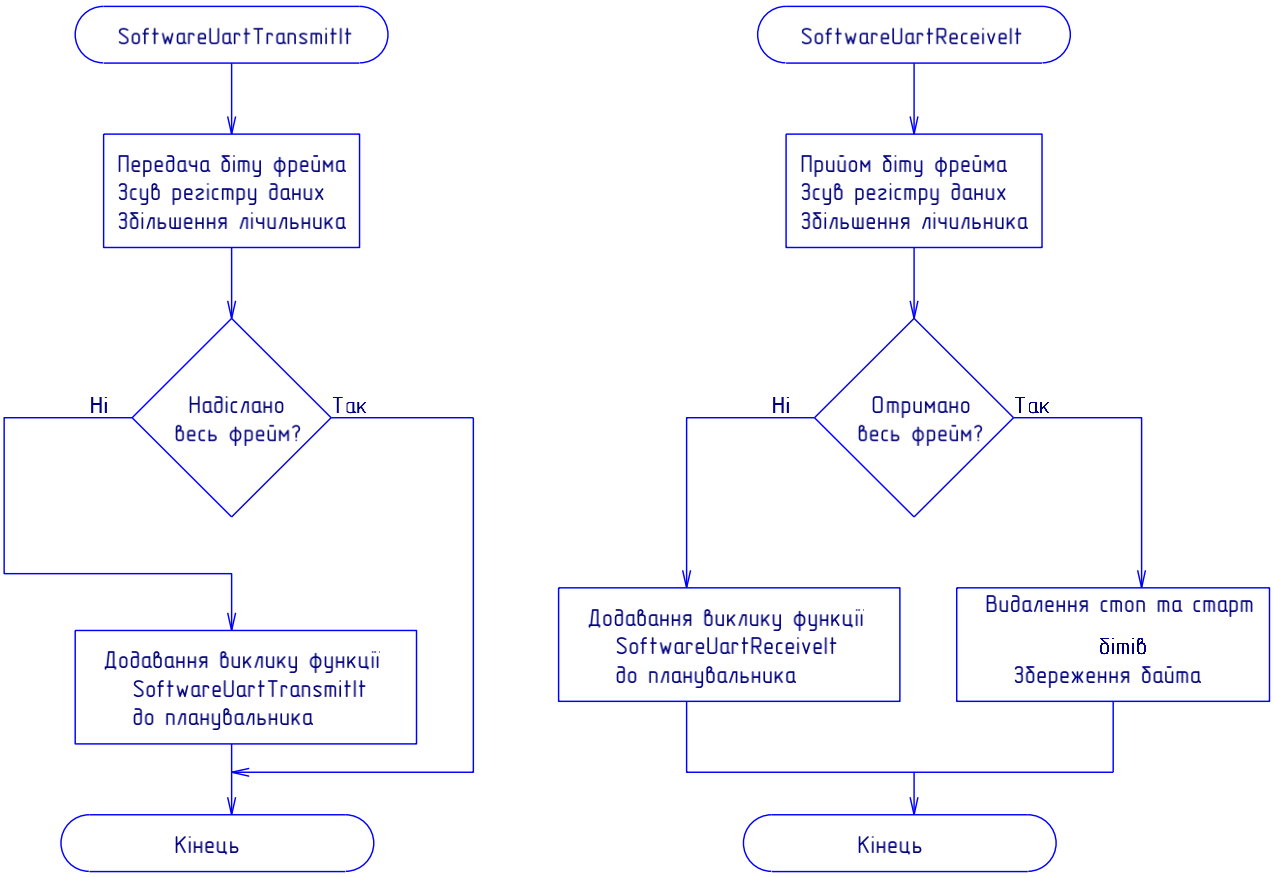


Рисунок 3.15 – Алгоритми роботи функцій SoftwareUartTransmitIt і SoftwareUartReceiveIt

Використання UART на перериваннях дає змогу вести багато потоковий та одночасний обмін. Складна реалізація потребує більше ресурсів процесора та може сповільнювати основну роботу програми, але є оптимальним у використанні при обміні на частотах до 9600бод.

4.ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В даному розділі розглядається процес передачі даних між пристроями по лінії живлення постійної напруги.

Розглянемо процес передачі даних від головного пристрою до відомого. Згідно з розробленого інтерфейсу передача даних відбувається за допомогою інтерфейсу UART, модулятора та демодулятора несучої частоти.

В рамках тестування використано зовнішній інтерфейс Bluetooth для передачі-прийому даних від смартфона до головного пристрою. Головний пристрій ретранслює ці дані по лінії живлення до відомого пристрою. Приймальна частина оброблює дані та формує пакет для передачі даних на вторинний пристрій відображення та контролю інформації (додатковий мікроконтролер). Отримані дані відображаються на семисегментному індикаторі.

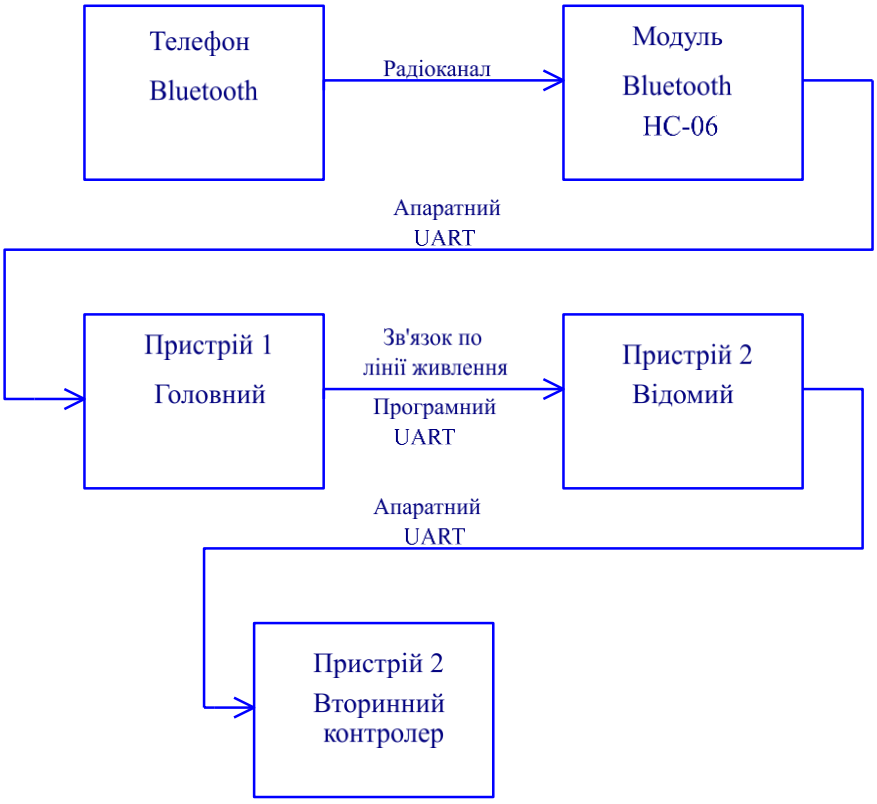


Рисунок 4.1 Структурна схема зв’язку

Початковою ланкою в передачі даних є смартфон. Ініціюємо передачу пакета «435\r\n» за допомогою модуля Bluetooth. Головний пристрій отримав пакет даних та ретранслював дані на лінію живлення до відомого пристрою.

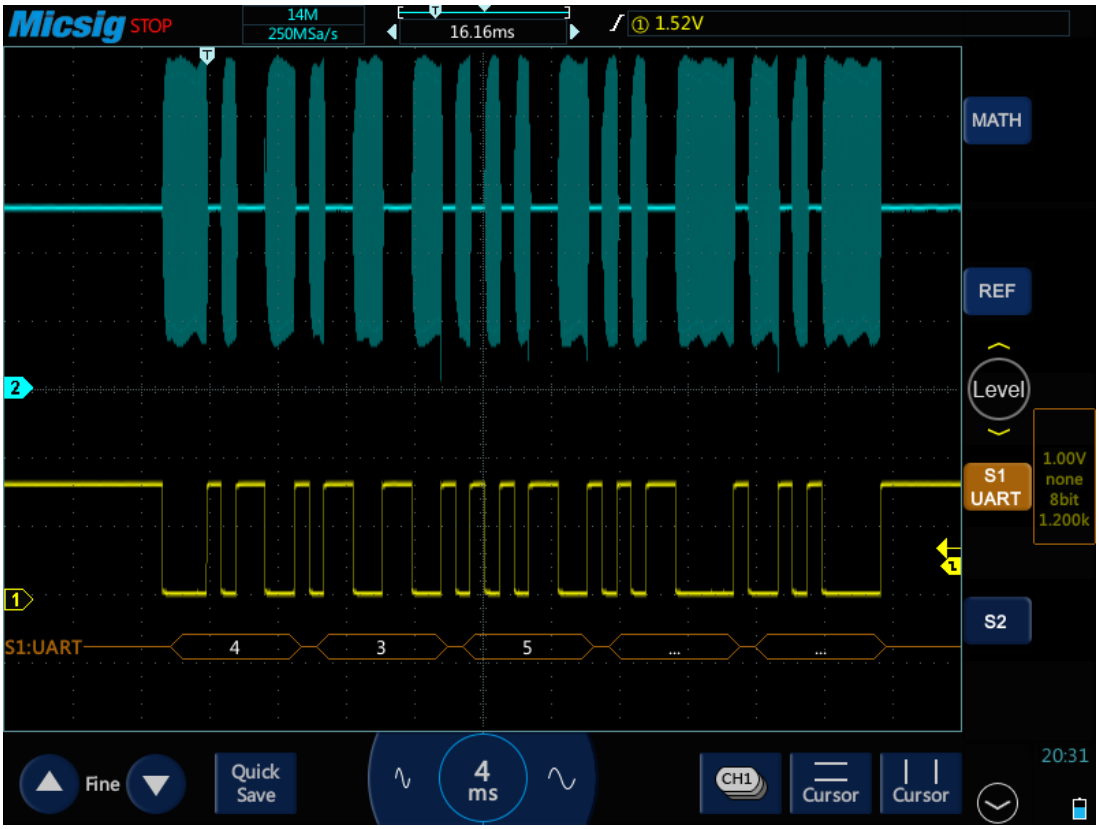


Рисунок 4.2 – Ілюстрація процесу демодуляції і відновлення даних

Осцилограма, що ілюструє процес демодуляції і відновлення даних. На цьому малюнку: дані з інтерфейсу UART(жовтий), модулюються поверх лінії живлення(синій), вбудований декодер даних зображує надісланий сигнал «435\r\n» (оранжевий).

Апаратна частина зв’язку пристроїв по лінії живлення модулює змінну напругу для передачі сигналу «нуль» 0. Передача сигналу «один» 1 є сигнал тиші – відсутність модуляції. Демонстративно зображено процес на рисунку 4.3.

Осцилограма демонструє процес початку отримання даних. На цьому малюнку : дані з інтерфейсу UART (жовтий) і модуляція на лінії живлення (синій). Як видно з осцилограми, модуляція на лінії живлення відбувається в момент коли інтерфейс UART передає «0» нуль.

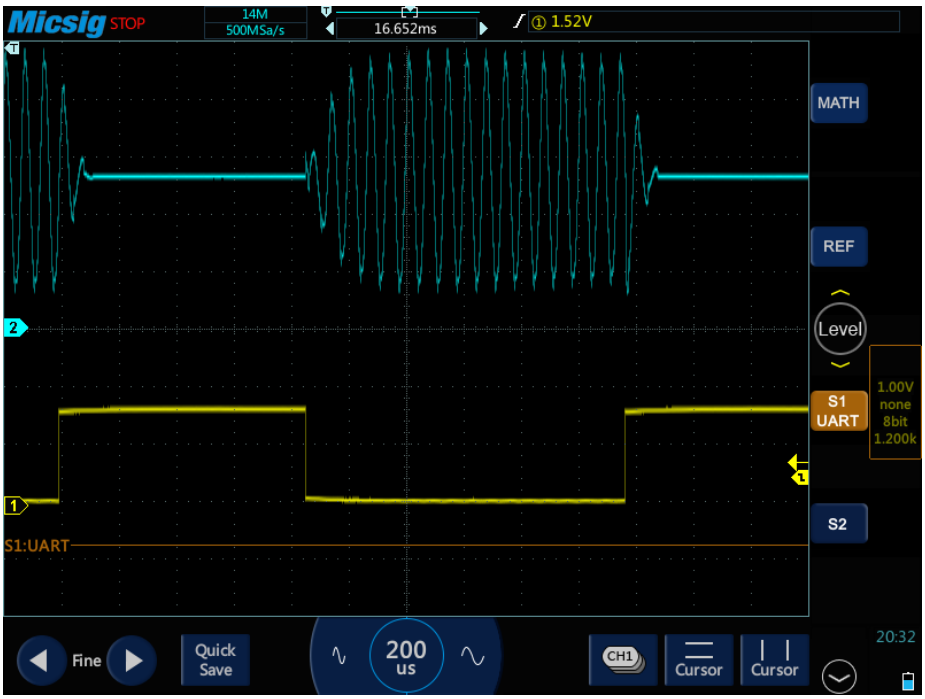


Рисунок 4.3 – Ілюстрація процесу початку отримання даних

Для зменшення завад модулюючий сигнал повинен мати якомога менше вторинних складових гармонік, тому при передачі даних по лінії живлення застосовуються модулюючі сигнали близькі до синусоїди.

Саме синусоїдальні сигнали менш вразливі до впливу паразитних параметрів з'єднувальних дротів, тому для передачі даних на довгі відстані в якості модулюючого сигналу використовується саме такий сигнал. На рис. 4.4 зображено осцилограму моделюючого сигналу.

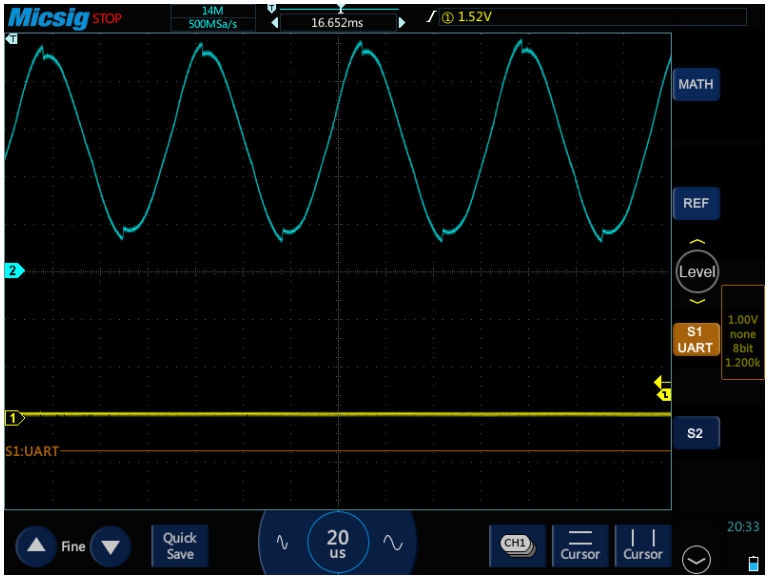


Рис.4.4 – Ілюстрація моделюючого сигналу

Робочий стенд тестування зображено на рис.4.5. На ньому зображено відомий пристрій(лівий верхній кут), ведучий пристрій(правий верхній кут), смартфон та Bluetooth модуль(правий нижній кут).

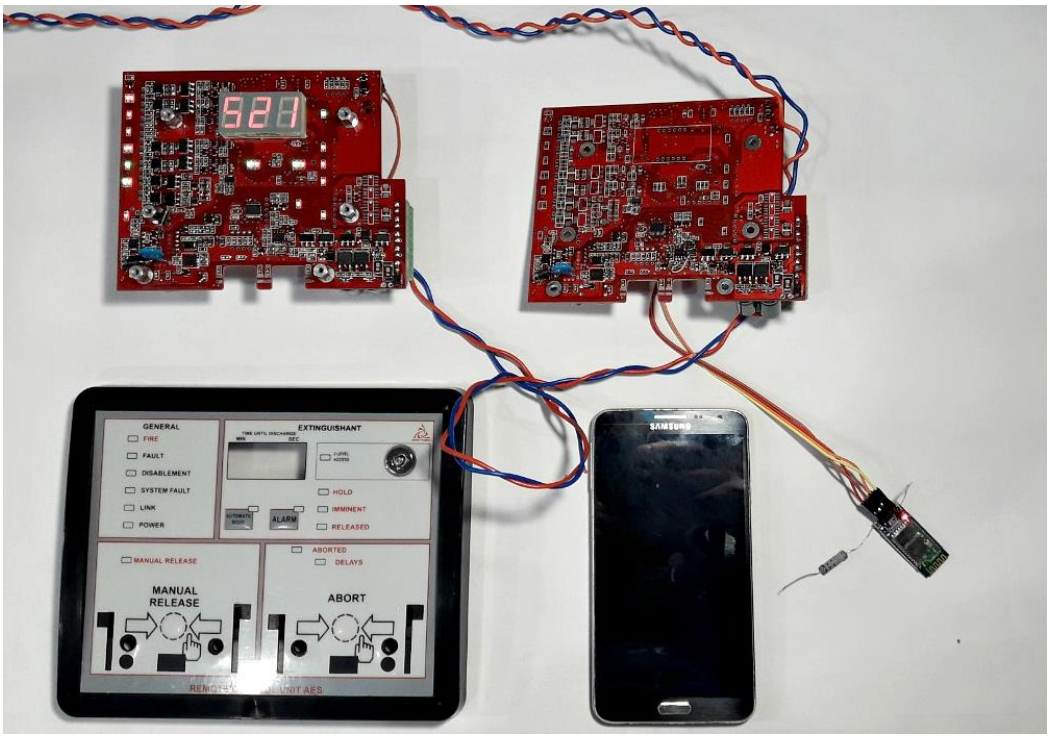


Рисунок 4.5 – Стенд тестування

В рамках тестування використано зовнішній інтерфейс Bluetooth для передачі-прийому даних від смартфона до головного пристрою. Головний пристрій приймає дані за допомогою апаратного UART.

Головний пристрій ретранслює ці дані по лінії живлення до відомого пристрою. Приймальна частина оброблює дані і реагує на команду, що прийшла з головного пристрою. У відповідь на ті чи інші команди буде відбуватися відображення даних на смисегментному індикаторі або будуть вмикатися або вимикатися світло діоди.

Для тестування обміну даними було створену просту процедуру яка обробляє прийняті байти:

```
1. sscanf(cmd_buf, "%s %d %d", &stringBuf, &value1, &value2);
2.
3.      if ( strcmp(stringBuf, "val") == 0) {
4.          SevenSegmentLedDisplay.Value = value1;
5.      }
6.      else if ( strcmp(stringBuf, "led") == 0) {
```

```
7.          Interface.Register2Data[value1] ^= 1<<value2;
8.      }
9.      else if ( strcmp(stringBuf, "dis") == 0) {
10.         interface.Register2Data[3] = 0;
11.         interface.Register2Data[4] = 0;
12.         interface.Register2Data[5] = 0;
13.     }
14.     else if ( strcmp(stringBuf, "en") == 0) {
15.         interface.Register2Data[3] = 0xff;
16.         interface.Register2Data[4] = 0xff;
17.         interface.Register2Data[5] = 0xff;
18.     }
```

Функція sscanf розпізнає отримані дані що знаходяться в масиві cmd_buf по наступному формату – строка, число, число, та записує їх у відповідні змінні stringBuf, value1, value2. В stringBuf зберігається основна команда – функція, а value1, та value2 її аргументи. В рамках тестування та налагодження процесу обміну даними створено наступні команди:

- Команда «val» записує в структуру для роботи з семи сегментним дисплеєм наступне трьохзначне число що буде виведено на екран.
- Команда «led» перемикає певний біт структури для роботи з світло діодами (вкл.\викл.)
- Команди «en» та «dis» вмикають чи вимикають усі світлодіоди відповідно.

На рис.4.6 зображено результат передачі команди «val 111». Відомий пристрій відображає на семисегментному індикаторі данні, що були відправлені зі смартфону.

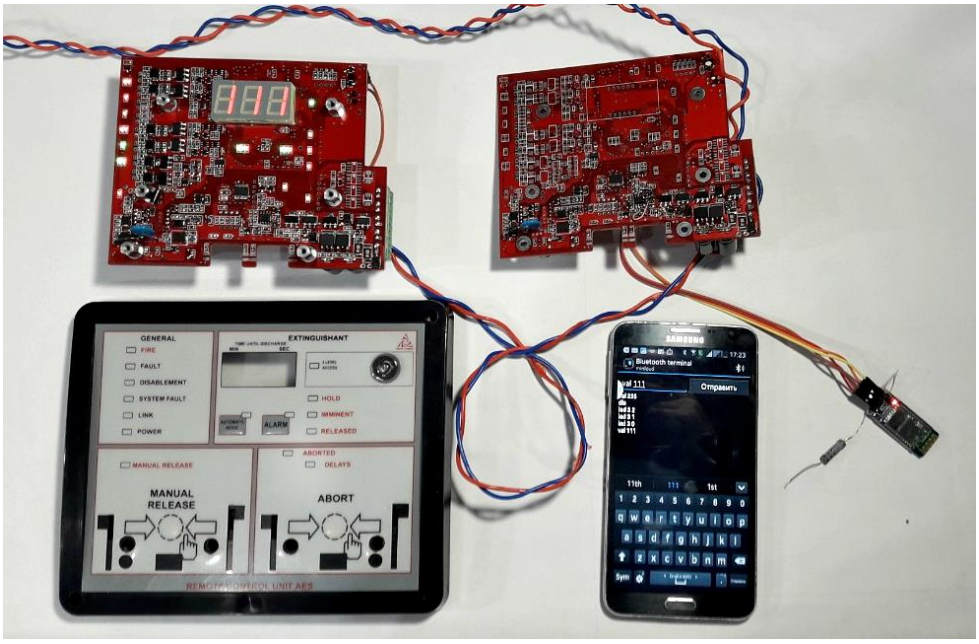


Рисунок 4.6 Відправка команди «val 111»

На рис.4.7 зображено результат передачі команди «dis», після цього усі світло діоди вимикаються.

Виходячи з отриманих результатів можна зробити висновок, що дані передаються коректно і даний метод передачі даних відповідає поставленим вимогам.

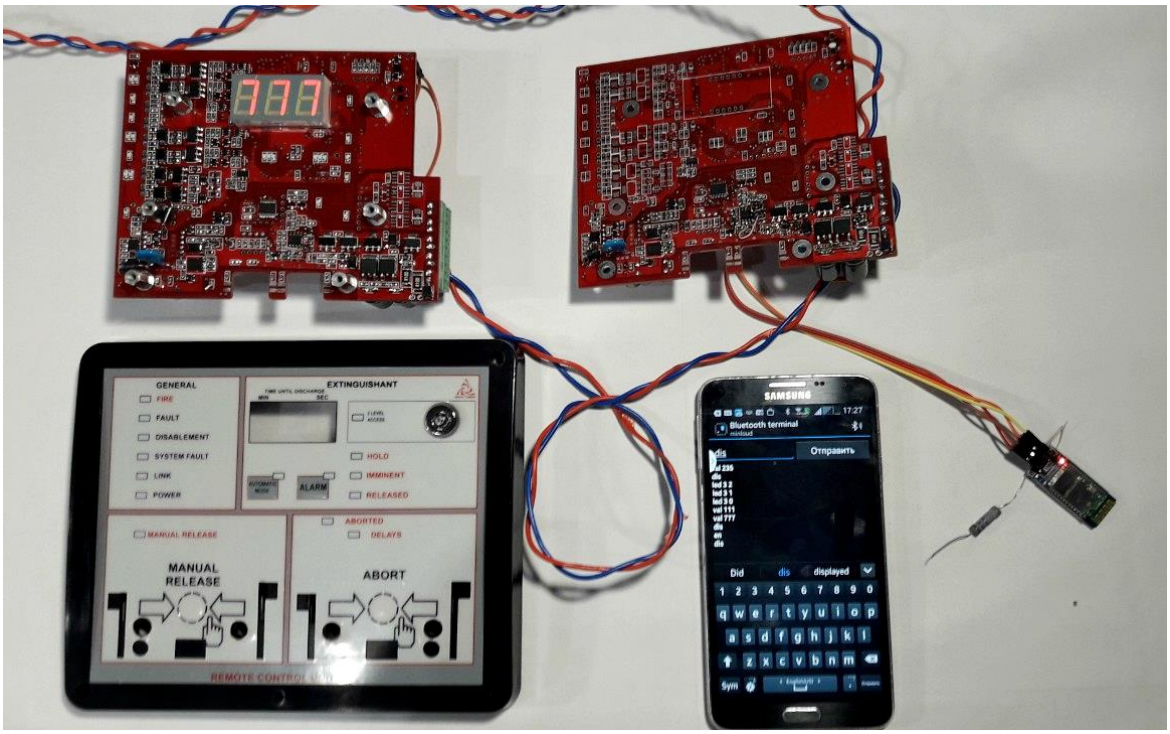


Рис.4.7 - Відправка команди «dis»

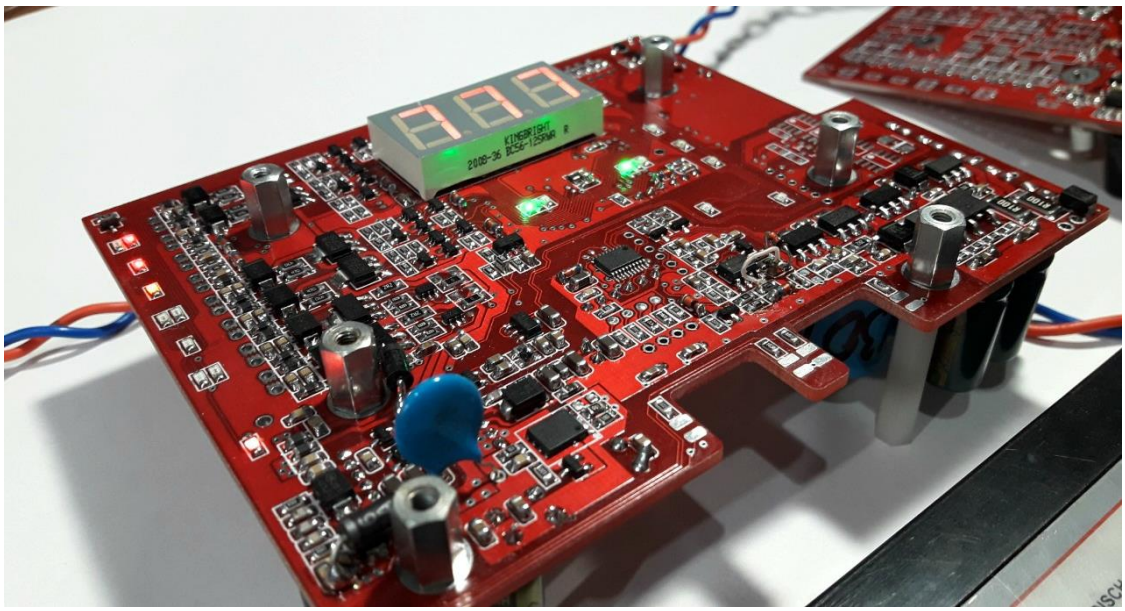


Рис.4.8 - Горіння світлодіодів після команди « led3 2»

ВИСНОВКИ

Розглянуті існуючі рішення створення мережевого зв’язку та рішення обміну даними дали можливість створення власної, децентралізованої та незалежної системи з високою стійкістю та простотою підключення мережі. Власна методика організації мережі підключення дозволяє уникнути появи випадків виходу з ладу системи внаслідок несправності одного з пристроїв системи чи лінії зв’язку та є безперечним рішенням для даної розробки. При цьому передача даних по лініях живлення тільки поліпшує якісні характеристики надійності цієї системи.

Організація зв’язку потребує додаткових вимог до програмного забезпечення. Значну частину роботи присвячено побудові складної програмної архітектури та реалізації програмних інтерфейсів зв’язку та виконано огляд рішень забезпечення цілісності даних використовуючи спеціалізовані промислові протоколи зв’язку. Через відсутність необхідної кількості апаратних модулів та інтерфейсів було розроблено складний механізм багатоканального одночасного прийому та обробки інформації. Реалізовано власний планувальних задач який забезпечує обробку даних в режимі реального часу. Розглянуто методи побудови багатозадачності та оглянуто існуючі операційні системи реального часу.

Виконано огляд програмних пакетів для розробки програмного забезпечення та бібліотек STM32 HAL. Саме ці бібліотеки дозволяють гнучко працювати з периферією мікроконтролера, мають високий рівень надійності та можливість обробки помилок підчас виконання програми.

Загалом, завдяки проведеній роботі було досягнуто значних успіхів у підвищенні якості та надійності системи пожежної безпеки, а сучасних підхід сприяє досягенню ведучих позицій на ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оліфер В. Г. Компьютерные сети. Принципы, технологии, протоколы. / В. Г. Оліфер, Н. А. Оліфер. – Питер: Питер, 2013. – 944 с.
2. Асмаков С. Интерфейс Bluetooth: разберемся с нюансами. / Сергій Асмаков. // КомпьютерПресс. – 2013. – №3. – С. 34–36.
3. Нікіфоров А.В. Технологія PLC – телекомунікація по мережам живлення // Мережі та системи зв'язку, 2002. – №5.
4. Elahi A. ZigBee Wireless Sensor and Control Network / A. Elahi, A. Gschwender. – Crawfordsville: Donneley && Sons, 2009. – 120 с.
5. Wang C. ZigBee Network Protocols and Applications / C. Wang, T. Jiang, Q. Zhang. – New York: CRC Press, 2014. – 345 с.
6. Warren G. Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC / Gay Warren. – Ontario: Apress, 2018. – 320 с.
7. Muhammad A. Stm32 Arm Programming for Embedded Systems / A. Muhammad, C. Shujen, G. Eshragh. – Ontario: LLC-Create Space, 2018. – 378 с.
8. Архітектура Mesh-мережі [Електронний ресурс] // lektsii.org. – 2015. – Режим доступу до ресурсу: https://lektsii.org/1-74362.html.
9. Архітектура ОС QNX [Електронний ресурс] // Архітектура ОС QNX. – 2016. – Режим доступу до ресурсу: http://um.co.ua/8/8-16/8-161106.html.
10. Попов Р. Микроконтроллеры STM32 «с нуля» [Електронний ресурс] / Роман Попов // КОМПЭЛ. – 2011. – Режим доступу до ресурсу: https://www.compel.ru/lib/ne/2011/2/4-mikrokontrolleryi-stm32-s-nulya.